Programming JavaServer Faces

Version 2.0.0 June 2012



Copyright © Descriptor Systems, 2012
All Rights Reserved.

This document may not be copied or reproduced in any form without prior written consent of Joel Barnum of Descriptor Systems.

All trademarks belong to their respective companies.

You can contact Descriptor Systems at:

http://www.descriptor.com
info@descriptor.com
319-362-3906
P.O. Box 461
Marion, IA 52302

Programming JavaServer Faces

Chapter 1: Introduction to the Course

JavaServer Faces Programming 1-1
Legal Information 1-2
JavaServer Faces Programming 1-3
Introductions 1-4
Course Description 1-5
Course Objectives 1-6
Sample Agenda 1-7
Sample Agenda, cont'd 1-8
Sample Agenda, cont'd 1-9
Course Logistics 1-10

Chapter 2: Introduction to JSF

Introduction to JSF 2-1
What is JSF? 2-2
JSF History 2-3
JSF Feature Overview 2-4
JSF Implementations 2-5
JSF Development Environments 2-6
JSF Development Process 2-7
JSF Hello, World: XHTML 2-8
JSF Hello, World: JSP 2-9
JSF Hello, World, cont'd 2-10
JSF Hello, World, cont'd 2-11
JSF Hello, World, cont'd 2-12
JSF Hello, World, cont'd 2-13
JSF Hello, World, cont'd 2-14
JSF Hello, World, cont'd 2-15
Chapter Summary 2-16

Chapter 3: Writing JavaBeans

Writing JavaBeans 3-1

What is a JavaBean? 3-2
Why Use JavaBeans? 3-3
JavaBean GUI Toolkits 3-4
JavaBeans versus Enterprise JavaBeans 3-5
JavaBean Requirements 3-6
JavaBean Features 3-7
Visual versus Non-Visual Beans 3-8
What are Reflection and Introspection? 3-9
Writing JavaBean Methods 3-10
JavaBean Properties 3-11
Simple Properties 3-12
Implementing Simple Properties 3-13
Calculated Properties 3-14
Accessing Simple Properties From a JSP 3-15
Indexed Properties 3-16
Quick Practice 3-17
Optional Coverage 3-18
Introduction to Events 3-19
Event Architecture 3-20
The Event Object Class 3-21
Event Listener Interfaces 3-22
Registering for an Event 3-23
Registering for an Event, cont'd 3-24
Firing an Event Based on Property Change 3-25
Firing an Event 3-26
Event Consumers 3-27
Bound Properties 3-28
Implementing a Bound Property 3-29
Implementing a Bound Property, cont'd 3-30
Listening to a Bound Property 3-31
Advanced JavaBean Techniques 3-32
Review Questions 3-33
Chapter Summary 3-34

Chapter 4: J2EE Review

JEE Review 4-1
What is Java Enterprise Edition? 4-2
JEE Application Servers 4-3
JEE Applications 4-4
The Model-View-Controller Architecture 4-5
Bean Scopes 4-6
JEE Packaging Model 4-7
Web Application Directory Structure 4-8
The Web Deployment Descriptor 4-9
Deployment Descriptor Elements 4-10
Forwarding versus Redirecting 4-11
Chapter Summary 4-12

Chapter 5: JSF Architecture

JSF Architecture 5-1
JSF Requirements 5-2
Configuring the Faces Servlet 5-3
Configuring the Faces Servlet, cont'd 5-4
JSF URLs 5-5
Optional: Preventing Direct Page Access 5-6
Optional: Providing a Start Page 5-7
JSF View Pages 5-8
JSF Namespaces 5-9
JSF JSP Tag Libraries 5-10
Sample JSF Page: XHTML 5-11
Sample JSF Page: JSP 5-12
Managed Beans 5-13
Introduction to the JSF Expression Language 5-14
Introduction to JSF Components 5-15
The Component Tree 5-16
Typical Request Processing 5-17
Invoking an Application Method 5-18
Accessing the Servlet Environment 5-19

Introduction to Navigation 5-20
Chapter Summary 5-21

Chapter 6: Request Processing and Navigation

Request Processing and Navigation 6-1
JSF Components 6-2
The Component Tree 6-3
Request/Response Lifecycle 6-4
Restore View Phase 6-5
View State 6-6
Apply Request Values Phase 6-7
Displaying Conversion Errors 6-8
Validation Phase 6-9
Update Model Phase 6-10
Invoke Application Phase 6-11
Renderer Response Phase 6-12
Backing Beans 6-13
JSF Navigation 6-14
Navigation Rule Syntax 6-15
Static Navigation 6-16
Dynamic Navigation 6-17
Using Redirection 6-18
Navigation Wildcards 6-19
Using a GUI Tool for Navigation Rules 6-20
Additional JSF2 Navigation Features 6-21
Implicit Navigation 6-22
Conditional Navigation 6-23
Chapter Summary 6-24

Chapter 7: Managed Beans and the JSF EL

Managed Beans and the JSF EL 7-1
What is a Managed Bean? 7-2
Using Managed Beans 7-3
Bean Annotations 7-4
Bean Scopes 7-5

Dependency Injection and Wiring....7-6
Injecting Simple Values....7-7
Injecting Lists....7-8
Injecting Maps....7-9
Wiring Bean References....7-10
Rules for Bean Scopes....7-11
JSF Expression Language....7-12
Why Another Expression Language?....7-13
JSF EL Operators....7-14
Operator Examples....7-15
JSF EL Implicit Variables....7-16
Implicit Variable Examples....7-17
Evaluating Expressions in Java....7-18
Chapter Summary....7-19

Chapter 8: JSF Components

JSF Components....8-1
JSF is a Component-Based Framework....8-2
JSF Provides a Pluggable Architecture....8-3
How Does JSF Create Component Objects?....8-4
The Client-Server Model....8-5
JSF Tags....8-6
Core Tags....8-7
Core Tags, cont'd....8-8
HTML Tags....8-9
HTML Tags, cont'd....8-10
Common Attributes....8-11
HTML Pass Through Attributes....8-12
HTML Output Tags....8-13
Output Tags Example....8-14
The h:form Tag....8-15
HTML Input Tags....8-16
Input Tags Example....8-17
HTML Command Tags....8-18

Command Tags Example 8-19
Single Selection Components 8-20
Single Selection Example 8-21
Multiple Selection Tags 8-22
Multiple Selection Example 8-23
Conditionally Enabling Components 8-24
Conditionally Displaying Components 8-25
Chapter Summary 8-26

Chapter 9: Conversion and Validation

Conversion and Validation 9-1
Data Conversion 9-2
JSF Request Lifecycle 9-3
Standard Converters 9-4
Converter Example 9-5
Converting Dates 9-6
Displaying Conversion Error Messages 9-7
Conversion Error Messages, cont'd 9-8
Custom Converter Error Messages 9-9
Custom Converter Error Messages, cont'd 9-10
Custom Converter Error Messages, cont'd 9-11
Standard Converter and Validation Messages 9-12
Custom Converters 9-13
Custom Converter Example 9-14
What is Validation? 9-15
Validation Scenarios 9-16
Converters vs Validators 9-17
Required Fields 9-18
JSF Standard Validators 9-19
Displaying Validation Error Messages 9-20
Custom Validation Error Messages 9-21
Bean Validation Framework 9-22
History of Bean Validation 9-23
Using Bean Validation 9-24

A Simple Example: The Bean....9-25
Bean Validation Annotations....9-26
Defining a Custom Validator....9-27
Custom Validator Example....9-28
Bypassing Conversion and Validation....9-29
Chapter Summary....9-30

Chapter 10: Event Handling

Event Handling....10-1
Event Handling in Java....10-2
JavaBean Events....10-3
JavaBean Event Architecture....10-4
JSF Event Categories....10-5
Action Events....10-6
Value Change Events....10-7
Immediate vs Non-Immediate Events....10-8
Configuring an Immediate Event....10-9
Immediate Event Example....10-10
Phase Listeners....10-11
Phase Listener Example....10-12
System Events....10-13
Global System Events....10-14
Chapter Summary....10-15

Chapter 11: Panels and Data Tables

Panels and Data Tables....11-1
What is a PanelGrid?....11-2
The h:panelGrid Tag....11-3
The h:panelGroup Tag....11-4
Introduction to Data Tables....11-5
The h:dataTable Tag....11-6
Data Table Iteration....11-7
Data Table Facets....11-8
Alternating Row Styles....11-9
Components in Data Table Cells....11-10

Editable Table Cells 11-11
Table Data Models 11-12
Using a Data Model 11-13
Advanced Data Table Topics 11-14
Using a Data Table with a Database 11-15
Accessing the Database 11-16
What is JPA? 11-17
Java Persistence Ancestry 11-18
Java Persistence Overview 11-19
JPA Data Table Example 11-20
Paging Large Result Sets 11-21
JPA Paging Support 11-22
Enabling and Disabling Paging Buttons 11-23
Paging Example 11-24
Chapter Summary 11-25

Chapter 12: JSF and Internationalization

JSF and Internationalization 12-1
What is Internationalization ? 12-2
JSE Support for I18N 12-3
What is Unicode? 12-4
Language and Country Codes 12-5
What is a Locale? 12-6
How Does JSF Determine Locale? 12-7
Selecting a Locale Explicitly 12-8
Message Bundle Property Files 12-9
Formatting Dates and Numbers 12-10
Complete I18N Example 12-11
Chapter Summary 12-12

Chapter 13: Introduction to Facelets

Introduction to Facelets 13-1
What is Facelets? 13-2
Why XHTML Instead of JSP? 13-3
Configuring the View File Extension 13-4

Writing a Facelets Application....13-5
Common Layout and Content....13-6
Writing a Facelets Template....13-7
Writing a Facelets Page....13-8
Another Facelets Page....13-9
The Facelets Namespace....13-10
The jsfc Attribute....13-11
Inlining Text....13-12
Facelets and JSTL....13-13
Chapter Summary....13-14

Chapter 14: Introduction to AJAX

Introduction to AJAX....14-1
What is AJAX?....14-2
Traditional Web Processing....14-3
AJAX Processing....14-4
AJAX Applications....14-5
Introduction to JavaScript....14-6
Defining JavaScript....14-7
JavaScript Object Syntax....14-8
JavaScript Functions....14-9
JavaScript Variables....14-10
JavaScript Events....14-11
JavaScript Events Example....14-12
Introduction to the HTML DOM....14-13
The DOM Tree....14-14
Accessing Nodes....14-15
Finding HTML Elements by ID....14-16
The XMLHttpRequest Object....14-17
Retrieving an XMLHttpRequest Reference....14-18
Sending a Request with an Asynchronous Response....14-19
Sending a Request with an Asynchronous Response, cont'd....14-20
Using a Servlet to Respond....14-21
Updating the Document....14-22

Complete Example 14-23
Complete Example Listing 14-24
Complete Example Listing, cont'd 14-25
Complete Example Listing, cont'd 14-26
Complete Example Listing, cont'd 14-27
Complete Example Listing, cont'd 14-28
Complete Example Listing, cont'd 14-29
Complete Example Listing, cont'd 14-30
Complete Example Listing, cont'd 14-31
Chapter Summary 14-32

Chapter 15: JSF and Ajax

JSF and Ajax 15-1
JSF and Ajax 15-2
JSF2 Ajax Support 15-3
The f:ajax Tag 15-4
Introduction to Partial Page Rendering 15-5
Partial Page Rendering 15-6
Partial Page Rendering Options 15-7
Ajax Requests Without Form Submission 15-8
Introduction to Execute Lists 15-9
Execute List Options 15-10
Rendering vs Executing 15-11
Using a JSF Phase Listener 15-12
Chapter Summary 15-13

Chapter 16: The OWASP Top Ten

The OWASP Top Ten 16-1
What is OWASP? 16-2
The OWASP Top Ten for 2010 16-3
Injection 16-4
Mitigating Injection in Java 16-5
Cross-Site Scripting 16-6
Mitigating XSS in Java Web Apps 16-7
Broken Authentication and Sessions 16-8

Mitigating Broken Sessions in Java Web Apps 16-9
Insecure Direct Object References 16-10
Mitigating Insecure Object Vulnerabilities 16-11
Cross-Site Request Forgery 16-12
CRSF Example 16-13
Mitigating CRSF 16-14
Mitigating CRSF, cont'd 16-15
Security Misconfiguration 16-16
Mitigating Security Misconfiguration 16-17
Insecure Cryptographic Storage 16-18
Mitigating Insecure Storage 16-19
Failure to Restrict URL Access 16-20
Mitigating Failure to Restrict URL Access 16-21
Insufficient Transport Layer Protection 16-22
Mitigating Insufficient Transport Layer Protection 16-23
Unvalidated Redirects and Forwards 16-24
Mitigating Unvalidated Redirects and Forwards 16-25
The OWASP Java Project 16-26
Chapter Summary 16-27

JavaServer Faces Programming

Version 2.0.0 June 2012



1 - 1

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Legal Information

Copyright (C) Descriptor Systems 2001, 2012

All Rights Reserved

All trademarks are owned by their respective companies

You can contact Descriptor Systems at:
http://www.descriptor.com
salesinfo@descriptor.com
319-362-3906

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaServer Faces Programming

- Introductions
- Objectives

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introductions

- Please introduce yourself:
  - Name:
  - Java Background:
  - JSP and Servlets Background:
  - JSF Background (if any):
  - Your goal for the course:

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Course Description

Description: This course covers how to write programs that use the JavaServer Faces 2.x framework.

Prerequisites: Basic Java and JEE Web application programming experience is assumed.

Audience: Business analysts, developers, managers and other people interested in learning how to program JSF.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Course Objectives

After taking this course, you will be able to:

- Understand the basic architecture of the JSF framework
- Understand how JSF applications process requests
- Write applications using JSF navigation
- Write managed bean classes and manipulate managed beans using the JSF expression language
- Check user input using JSF validators
- Use JSF components for input, output and display of tabular data
- Internationalize JSF applications
- Understand the basics of the Facelets view technology
- Write JSF applications that use Ajax

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda

```
Day 1
-------

Introduction to the Workshop

Introduction to JSF
  Lab 1: Intro to JSF

JavaBeans

JEE Web Application Review

JSF Architecture
  Lab 2: JSF Architecture

Requst Processing and Navigation
  Lab 3: Navigation

Managed Beans and the JSF EL
  Lab 4: Beans and EL
```


Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda, cont'd

```
Day 2
-------

JSF Components
  Lab 5: Components

Conversion and Validation
  Lab 6: Conversion and Validation

Event Handling
  Lab 7: Events

Panels and Data Tables
  Lab 8: Panels and Data Tables
```


Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample Agenda, cont'd

```
Day 3
-----

Internationalization
  Lab 9: I18N

Facelets
  Lab 10: Facelets

Introduction to Ajax

JSF and Ajax
  Lab 11: JSF and Ajax

The OWASP Top Ten
  Lab: Case Study
```


Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Course Logistics

- Schedule
    - Start time
    - Lunch
    - End time
    - Breaks

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JSF

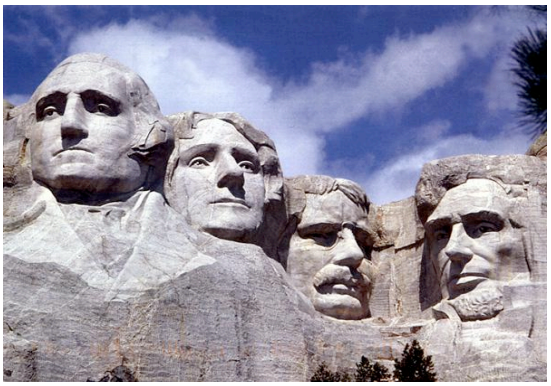
- What is JSF?
- JSF Hello, World

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is JSF?

- JavaServer Faces is a **component-based** framework for developing Java-based Web applications
- JSF supports a "rapid application development" programming style, especially coupled with a visual development tool
- JSF is a standard part of Java Enterprise Edition (JEE)





A framework is a set of prewritten code that you extend to create your own applications. JSF differs from other JEE Web frameworks like Apache Struts in that it's component-based, which means the framework provides, and encourages the use of components, such as buttons, lists and so forth. In contrast, Struts is more of an "action" framework where the focus is on handling Web requests.

JSF borrows inspiration from client-side user-interface systems like Java's Swing, where you also build "screens" from components.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF History

- JSF was defined through the Java Community Process as JSR 127
- Version 1.0 of the JSF specification was released in March 2004
- Version 2.0 was released in June of 2009
- Version 2.1 was released in October of 2010

There are some significant updates in Version 2 over Version 1, mostly involving annotations and dependency injection.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Feature Overview

- A set of pre-written components (e.g. buttons, entry fields, data tables)
- A standard component programming model that encourages third-party component development
- An event-driven programming model similar to that of Java Swing
- Facilities for validating user input
- A page navigation model
- Support for internationalization (I18N)
- "Pluggable" view technologies (e.g. JSP or Facelets)
- An expression language for working with "managed beans"

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Implementations

- The Sun JSF Reference Implementation (a.k.a. Mojarra)
- An implementation provided by a JEE application server
- Apache MyFaces

JSF is a specification, i.e. a document. Here we show a couple of implementations of the JSF specification.

You can find out more about Mojarra at http://javaserverfaces.java.net.

You can find out more about MyFaces at http://myfaces.apache.org.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Development Environments

- Simple text editor and command-line tools
- Java-based IDE such as Eclipse
- JSF-aware IDE such as IBM RAD or Sun NetBeans

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Development Process

- Write the view pages (XHMTL or JSP) that define the user interface
- Code the **managed beans** that store the user-entered data
- Write the faces-config.xml configuration file
- Configure the JEE Web deployment descriptor for the Faces servlet

Facelets is a JSF view technology that was an extra add-on to JSF in version 1, but is included with JSF 2.

"Managed beans" are simply JavaBean-like objects (POJOs) that you configure in JSF so that the JSF runtime manages their lifecycle.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Hello, World: XHTML

```
1    <html xmlns="http://www.w3.org/1999/xhtml"
2       xmlns:h="http://java.sun.com/jsf/html">
3      <h:head>
4        <title>What's Your Name?</title>
5       </h:head>
6      <h:body>
7        <h:form>
8            <h3>JSF Hello, World</h3>
9            <p>
10             Please enter your name:
11             <h:inputText value="#{hello.name}" />
12           </p>
13           <p>
14             <h:commandButton value="Submit"
15                action="#{hello.mymethod}" />
16           </p>
17       </h:form>
18     </h:body>
19   </html>
```

Using XHTML files for the view is recommended in JSF2 and later.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Hello, World: JSP

```
1    <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
2    <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
3    <html>
4      <f:view>
5        <body>
6          <h:form>
7            <h3>JSF Hello, World</h3>
8            <p>
9              Please enter your name:
10             <h:inputText value="#{hello.name}" />
11           </p>
12           <p>
13             <h:commandButton value="Submit"
14                action="#{hello.mymethod}" />
15           </p>
16         </h:form>
17       </body>
18     </f:view>
19   </html>
```


Using JSPs for the view is supported in all versions of JSF, but for JSF 2 and later, it's recommended to use XHTML files instead.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Hello, World, cont'd

```
1    package hello;
2
3    @ManagedBean(name = "hello")
4    @SessionScoped
5    public class HelloBean
6    {
7        private String name;
8
9        public String getName()
10       {
11           return name;
12       }
13
14       public void setName(String name)
15       {
16           this.name = name;
17       }
```

2 - 10

Here we show the start of our "managed bean" that stores the data that the user enters into the form. This bean also processes the request when the user presses the Submit button (shown on next page).

Note that the annotations, @ManagedBean and @SessionScoped, are only supported in JSF2 and later.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Hello, World, cont'd

```
18        public String mymethod()
19        {
20            if (name.equalsIgnoreCase("SAM"))
21                return "success";
22            else
23                return "bad-name";
24        }
25    }
```


Finishing the listing of HelloBean.java, here we show the "action" method that the Faces controller invokes when the user presses the form's Submit button. Note how we return a String that indicates which action to take next.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Hello, World, cont'd

```
1     <!DOCTYPE faces-config PUBLIC
2       ...>
3
4     <faces-config>
5       <navigation-rule>
6         <from-view-id>/index.jsp</from-view-id>
7         <navigation-case>
8           <from-outcome>success</from-outcome>
9         <!--  <to-view-id>/hello.jsp</to-view-id> -->
10          <to-view-id>/hello.xhtml</to-view-id>
11        </navigation-case>
12        <navigation-case>
13          <from-outcome>bad-name</from-outcome>
14        <!--  <to-view-id>/badname.jsp</to-view-id> -->
15          <to-view-id>/badname.xhtml</to-view-id>
16        </navigation-case>
17      </navigation-rule>
```

Here we start the listing of the faces-config.xml file that configures our Faces application. We start by defining the navigation rules -- for example, if the controller method returns "success", Faces will transfer control to "hello.jsp".

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Hello, World, cont'd

```
18      <!-- only needed if annotations not used in
19           managed bean source file -->
20      <managed-bean>
21        <managed-bean-name>hello</managed-bean-name>
22        <managed-bean-class>
23           hello.HelloBean
24        </managed-bean-class>
25        <managed-bean-scope>
26          session
27        </managed-bean-scope>
28      </managed-bean>
29    </faces-config>
```

Continuing the faces-config.xml file, here we show defining the "managed bean" that contains the get/set methods for the input properties as well as the controller method that processes the request.

Note that if we used the @ManagedBean and @SessionScoped annotations in HelloBean.java, then we can omit this configuration.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Hello, World, cont'd

```
1    <html xmlns="http://www.w3.org/1999/xhtml"
2       xmlns:h="http://java.sun.com/jsf/html">
3      <h:head>
4        <title>Hello</title>
5       </h:head>
6       <h:body>
7          <p>
8            Hi there, <h:outputText value="#{hello.name}"/>
9          </p>
10      </h:body>
11   </html>
```


Here we show the "hello.xhtml" file that's invoked if the controller method returns "success". Note how it can access the backing bean's "name" property using a JSF expression.

The "badname.xhtml" file is similar and not shown here.

Note that in JSF 1, this would be a JSP rather than an XHTML file.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Hello, World, cont'd

```
1    <?xml version="1.0" ?>
2    <!DOCTYPE web-app PUBLIC ...>
3    <web-app>
4      . . .
5      <servlet>
6        <servlet-name>Faces Servlet</servlet-name>
7        <servlet-class>
8          javax.faces.webapp.FacesServlet
9        </servlet-class>
10       <load-on-startup>1</load-on-startup>
11     </servlet>
12     <servlet-mapping>
13       <servlet-name>Faces Servlet</servlet-name>
14       <url-pattern>*.faces</url-pattern>
15     </servlet-mapping>
16   </web-app>
```

Here we show the web.xml deployment descriptor for our Faces application. Note how it defines the Faces servlet.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned

- The basics of JSF

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing JavaBeans

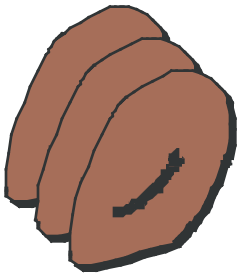
- What is a JavaBean?
- Properties, Methods and Events

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a JavaBean?

- A JavaBean is a Java class that is designed to manipulated by tools or by other programs (.e.g. JavaServer Pages) in a standard fashion
- The JavaBean specification defines the JavaBean as the standard component architecture for Java



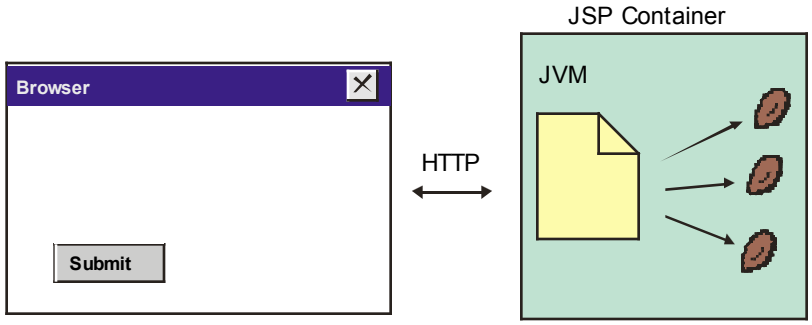
The JavaBean specification is available from:

http://java.sun.com/products/javabeans/docs/spec.html

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Why Use JavaBeans?

- Some development tools use JavaBeans to let you drag and drop to create graphical user interfaces (GUI)
- JavaServer Pages and other J2EE technologies can access JavaBeans that provide state and behavior

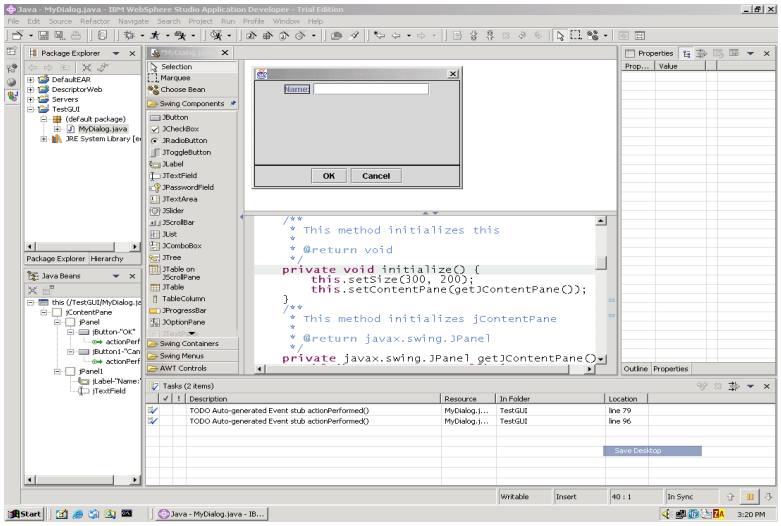


The JavaBean specification was initially created to facilitate GUI builders for rich clients, but since then, JavaBeans have more widely been used in server-side applications.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaBean GUI Toolkits

- IBM WebSphere Rational Application Developer
- Sun NetBeans
- Eclipse with Visual Editor Project Plugin
- Oracle JDeveloper
- JetBrains IntelliJ IDEA



Sun also provides the free Bean Builder tool that is a demonstration program on how to use JavaBeans to build GUIs.

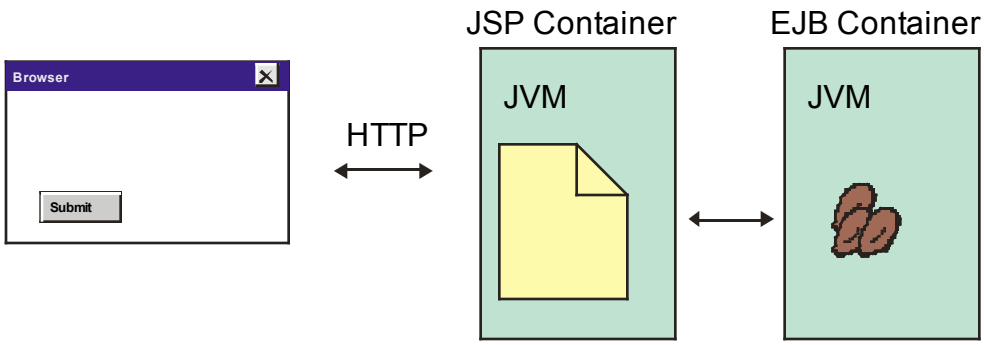
The Bean Builder is available from:

https://bean-builder.dev.java.net/

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaBeans versus Enterprise JavaBeans

- Enterprise JavaBeans implement specific interfaces so they can run within an **EJB container** that provides services such as transactions, security and persistence
- EJBs can be invoked remotely, while "normal" JavaBeans run in the same JVM as the caller



3 - 5

The EJB specification is an integral part of J2EE and allows for components that execute remotely. We will not cover EJBs in this class, but you should be aware that it's very possible to design web applications that include JSPs, servlets and EJBs. Such designs are complex but scalable, since they let you separate the business logic onto remote servers. In addition, since the EJB container provides powerful services, applications that use EJBs can have truly enterprise-class robustness.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaBean Requirements

- A JavaBean is a Java class that follows a few simple conventions
- To be a proper JavaBean, a Java class should:
  - Implement the Serializable interface
  - Be in a named package
  - Expose properties, methods and events as necessary
  - Define a no-argument constructor
  - Optionally packaged into a JAR file

This page lists some of the conventions you should follow to create compliant JavaBeans. We will see a complete example in a few pages.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaBean Features

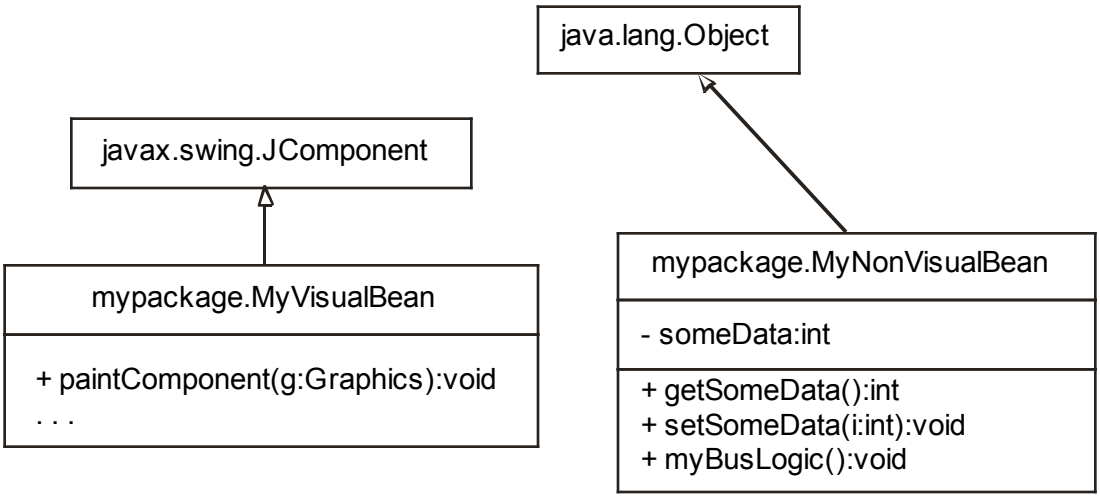
- A JavaBean can expose three kinds of features:
  - **Properties**, which are defined using get/set methods
  - **Methods**, which expose behavior
  - **Events**, which let the Bean call back to the outside world

The JavaBean specification provides precise definitions of how to code a JavaBean that provides these three kinds of features.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Visual versus Non-Visual Beans

- Visual beans are designed to appear in a graphical user interface (GUI)
- Non-visual beans generally define business logic and data



The Java class library defines many visual beans (e.g. JButton, JTextField and so forth) that you can use "out-of-the-box" and you can also create your own custom visual beans that integrate into GUI builders.

Non-visual beans generally contain business methods, data (properties) and perhaps fire events. Most GUI builders supply a mechanism so that GUIs can access non-visual beans.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What are Reflection and Introspection?

- The Java API includes a technique known as **reflection** that lets a program enumerate the public features of any Java class
- If the Java class follows some simple naming conventions, the examining program can make assumptions about the class using a process referred to as **introspection**
- For example, if a JavaBean has a two public methods, one named *getAddress* and one named *setAddress*, we can infer that the Bean defines a property named address

---

For example, a JSP container uses reflection and introspection to determine a Bean's features and thus check that the JSP author accesses the features correctly.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing JavaBean Methods

- A JavaBean method is simply a public method that exposes a behavior

```
1    public boolean calcHonorRoll()
2    {
3        if ( getGpa() > 3.5 )
4            return true;
5        else
6            return false;
7    }
```


This JavaBean method is taken from a StudentBean class.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaBean Properties

- A **property** exposes state from a JavaBean
- There are four kinds of properties defined in the JavaBean specification:
    - Simple Properties
    - Indexed Properties
    - Bound Properties (covered later)
    - Constrained Properties (not covered in this chapter)

Properties represent the data of a JavaBean and are probably the most-used feature of JavaBeans.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Simple Properties

- A **property** is defined by the existence of "getter" and "setter" methods
- Note that the property name is case sensitive!
- A **readonly property** defines only a "getter" method

```
1    public String getAddress() {...}
2    public void setAddress (String s) {...}
3    public boolean isSalaried() {...}
4    public void setSalaried(boolean b) {...}
5    public int getSerialNumber () {...}
```

Property Name	Property Type
address	String
salaried	boolean
serialNumber	int

Tools can introspect JavaBeans looking for methods that match a naming convention to infer properties. Many JavaBean tools will display the properties in some sort of a "property sheet" that allows a developer to view and modify the properties at design time.

This code snippet shows a fragment of a Bean class that defines three methods. The combination of the "getAddress" and "setAddress" methods implies a property named "address", while the existence of the "getSerialNumber" method implies a read-only property named "serialNumber".

Note that property names always begin with a lower-case character even though the method names include an upper-case character.

Also note that there's a special case for properties of type boolean: the "getter" method may be prefixed with "is" rather than "get", for example, "isHonorRoll()".

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Implementing Simple Properties

- Simple properties are often implemented by fields

```
1    package company;
2
3    public class Employee implements java.io.Serializable
4    {
5      private int serial_number;
6      private String emp_address;
7      private boolean exempt;
8
9      public void setSerialNumber(int n)
10     {
11        serial_number = n;
12     }
13     public int getSerialNumber()
14     {
15        return serial_number;
16     }
17     . . .
18   }
```

Note that the field name is pretty much irrelevant. It's the names of the "get/set" methods that determine the property name.

Also note that many popular IDEs can generate the "get/set" methods for you if you write the fields.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Calculated Properties

- Some simple properties are **calculated properties**; these are generally read-only

```
1    package company;
2
3    public class Employee
4      implements java.io.Serializable
5    {
6       private double salary;
7       private boolean exempt;
8
9       public double getFicaTax()
10      {
11         if (!exempt)
12           return salary * 0.12;
13         else
14           return salary * 0.14;
15      }
16      . . .
17   }
```

The read-only property exposed by this example is of type "double" and is named "ficaTaxRate".

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Accessing Simple Properties From a JSP

- A common use of JavaBeans is in Web applications that use the **Model-View-Controller** architecture
- In MVC, a servlet (the Controller) creates and intializes beans (the Model) and stores them so a JSP (the View) can access the beans' properties

displayemployee.jsp

```
1    <!-- assume servlet stored Employee bean
2         reference using name "theEmp" -->
3
4    <p>The employee's ID: ${theEmp.serialNumber}</p>
5    <p>The employee's address: ${theEmp.address}</p>
6    <p>Is it a salaried employee? ${theEmp.salaried}</p>
```

The special syntax in a JSP to access beans is referred to as the "JSP Expression Language".

Note that in the JSP EL, we access properties by name without explicitly calling the "get" method. And note that the property names begin with a lower-case character.

JSPs are supposed to be able to be written by HTML authors that don't necessarily know Java. The JSP EL "hides" the Java syntax, making it easier to access beans.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Indexed Properties

- **Indexed** properties are like simple properties accessed as an array

```
1    private String[] degrees = new String[4];
2
3    public String getDegree(int index)
4    {
5        return degrees[index];
6    }
7    public void setDegree(int index, String degree)
8    {
9        degrees[index] = degree;
10   }
```

Indexed properties are somewhat rarely used, but can be convenient. Here we show an indexed property named "degree" of type String. Note how the getter and setter methods accept the index.

In this example, we implemented the indexed property using an array, but that's not required (we could've used a collection class such as ArrayList instead). The important thing is that the get/set methods follow the naming pattern.

Also note that you can optionally provide methods that get/set the entire array, for example, String[] getDegrees().

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Quick Practice

- Is this a proper JavaBean? What are its property names?

```
1    package univ;
2
3    public class Student
4      implements java.io.Serializable
5    {
6       private double gradepoint;
7       private int student_identifier;
8
9       public int getId() {return student_identifier;}
10      public void setId(int n) {student_identifier=n;}
11
12      public double getGpa() {return gradepoint;}
13      public void setGpa(double d) {gradepoint=d;}
14
15      public boolean isHonorRoll() {return true;}
16   }
```


Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

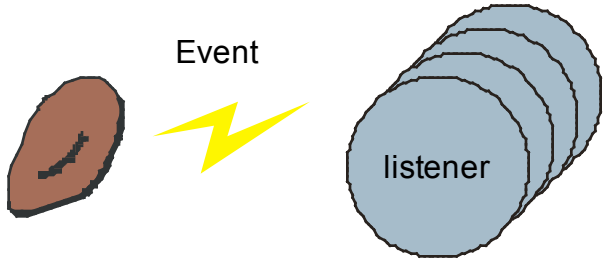
Optional Coverage

The rest of this chapter covers advanced JavaBean techniques

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Events

- Events allow JavaBeans to notify listeners of significant occurrences
- The following artifacts comprise an event:
    - An event-object class
    - An event listener interface
    - Register/deregister methods on the bean
    - Firing the event



Events are almost the opposite of methods -- methods let the outside world invoke behavior on the bean, while an event is the bean invoking behavior on "listeners", which are objects that have registered an interest in the event.

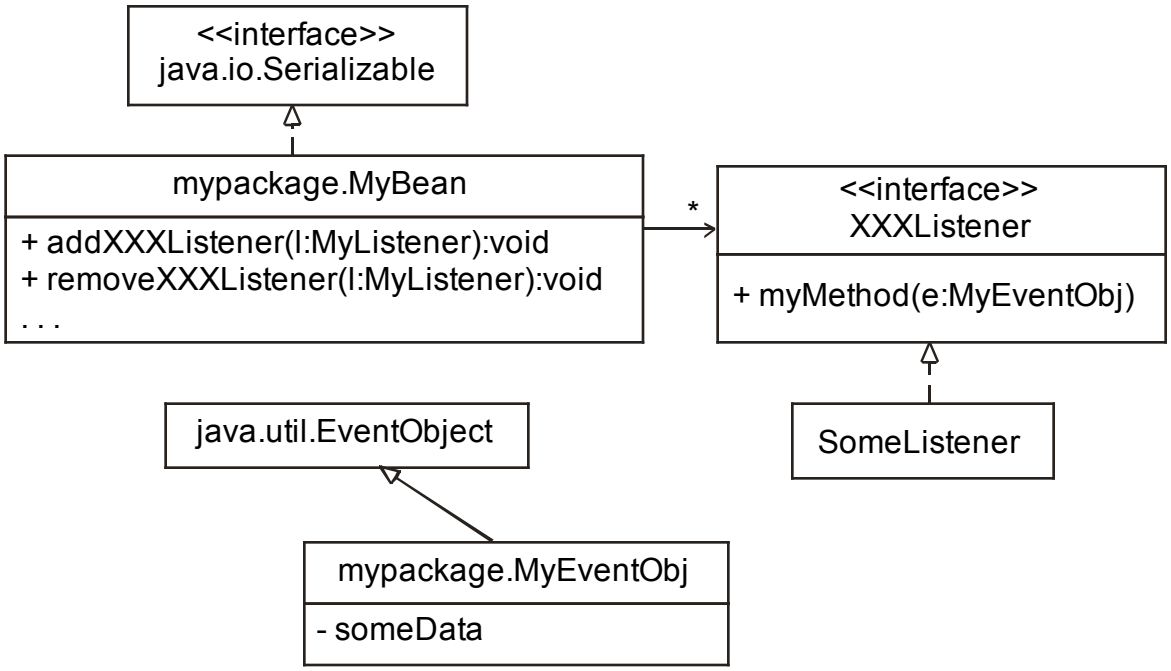
Events are especially useful in GUIs, where a user-interface component, say a button, notifies another object when the user clicks the button.

Note that there can be multiple listeners on a given event. The bean notifies all registered listeners when the event occurs.

JavaBean events are a variation on the Observer Design Pattern from the Design Patterns book by Gamma, et. al.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Event Architecture



The idea is that the bean maintains a list of interested listeners, each of which is typed to the interface that we define. The bean provides two methods: one to add a listener to the list, another to remove. Note that the names of these two methods are very significant -- we use introspection to infer the event name from the method names. In other words, whatever you put as the 'XXX' placeholder above will be interpreted as the event name.

We also can define a class that describes the event -- note that we pass an instance of the event class in the interface methods. This is optional -- if you don't have any extra information to pass, you can use the java.util.EventObject class directly.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Event Object Class

- Event objects carry information about the event

```
1    public class FlunkEvent extends java.util.EventObject
2    {
3        private String courseName;
4
5        public FlunkEvent(Object src, String courseName)
6        {
7            super(src);
8            this.courseName = coursName;
9        }
10       public String getCourseName()
11       {
12           return courseName;
13       }
14       public void setCourseName(String courseName)
15       {
16           this.courseName = courseName;
17       }
18   }
```

The Java class library (especially in the java.awt.event package) defines several event-object classes and you can write your own as we did here.

The standard EventObject superclass basically wraps a reference to the JavaBean that fired the event.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Event Listener Interfaces

- An event is essentially a callback from the JavaBean to the interested party
- The interested party must therefore define method(s) that the JavaBean can call
- We enforce this requirement using listener interfaces

```
1    public interface FlunkListener
2      extends java.util.EventListener
3    {
4      public void studentFlunked(FlunkEvent evt);
5    }
```


The listener interface defines the behavior(s) that any listener must provide.

Generally, the interface methods will take an event object as described on the last page.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Registering for an Event

- The JavaBean provides methods to allow interested listeners to add or remove themselves from the JavaBean's list of listeners
- It's the existence of the add/remove methods that allow tools to infer the event

These methods imply…	an event named:
addFlunkedOutListener() removeFlunkedOutListener()	FlunkedOut

Tools that examine JavaBeans use reflection and introspection to make inferences based on naming patterns. According to the JavaBean specification, if a bean provides methods named addXXXListener and removeXXXListener, we can infer an event named XXX.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Registering for an Event, cont'd

```
1    public class Student implements Serializable
2    {
3    . . .
4        private Vector listeners = new Vector();
5
6        public synchronized void
7          addFlunkedOutListener (FlunkListener fl)
8        {
9            listeners.add(fl);
10       }
11       public synchronized void
12         removeFlunkedOutListener(FlunkListener fl)
13       {
14           listeners.remove(fl);
15       }
16   . . .
17   }
```

The code to implement registering and de-registering listeners is trivial. Note, however the type of the arguments -- it's the type of the listener interface!

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Firing an Event Based on Property Change

- It's common for a JavaBean to fire an event based on the value of a property change
- This is similar, but a bit different than a **bound** property which fires an event regardless of the property value

```
1    public void setGpa(double gpa)
2    {
3       this.gpa = gpa;
4
5       if (gpa < 1.0)
6         fireFlunkedOutEvent ("Java 101");
7    }
```


Here we show an example JavaBean that fires the FlunkedOut event based on the GPA value, invoked the method shown a couple of pages ago.

We will cover bound properties in a moment, but note that this code only fires the event if the GPA is less than 1.0, while a bound property fires an event on any change to the property.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Firing an Event

- When the significant occurrence happens, the JavaBean **fires** the event by traversing the listener list, calling the interface method(s) on each listener

```
1    protected void fireFlunkedOutEvent(String course)
2    {
3        Vector copy = null;
4        FlunkEvent evt = new FlunkEvent(this,course);
5
6        synchronized(this)
7        {
8            copy = (Vector)listeners.clone();
9        }
10       Iterator iter = copy.iterator();
11       while (iter.hasNext())
12       {
13        ((FlunkListener)iter.next()).studentFlunked(evt);
14       }
15   }
```

The code shown here first copies the listener list in an atomic block (so that we don't get messed up if a listener attempts to add or remove itself while we're firing the event). Then it's a simple matter of traversing the list, calling the event method on each listener.

Of course, we need to call this method at some point -- that's sometimes the tricky part -- figuring out when to actually fire the event. We'll look at that in more detail for the Student bean in a moment.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Event Consumers

- Objects that are interested in the event must implement the event-listener interface and then register for the event

```
1    public class RespondToFlunk implements FlunkListener
2    {
3        public void studentFlunked(FlunkEvent evt)
4        {
5            System.out.println("Flunking: " +
6                    evt.getCourseName());
7        }
8
9        public static void main(String[] args)
10       {
11           RespondToFlunk me = new RespondToFlunk();
12           Student s = new Student();
13           s.addFlunkedOutListener(me);
14           s.setGpa(0.4);
15       }
16   }
```

Here we show a simple class that demonstrates listening to an event. The class implements the listener interface and then registers an object. Then, when we set the GPA to a low value, the JavaBean will fire the FlunkedOut event.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Bound Properties

- A **bound** property fires the predefined **PropertyChange** event when the property changes to notify interested listeners
- The PropertyChangeEvent object encapsulates the property name and the previous and new values of the property
- The java.beans.PropertyChangeSupport class makes implementing bound properties easy

A bound property fires an even whenever the property changes. This is especially handy in GUI applications where you want a segment of the user interface to change when some other segment changes. For example, consider a pair of list boxes, one of which shows a list of managers, while the other shows the list of subordinates of the currently selected manager.

If a given JavaBean has multiple bound properties, it fires the same event whenever any property changes -- note that the PropertyChangeEvent object includes the property name -- that allows listeners to determine which property changed.

Also note that the PropertyChangeEvent object includes the previous property value -- that helps listeners to decide whether or not to take action based on the change.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Implementing a Bound Property

```
1    public class Student implements Serializable
2    {
3        private String major;
4        . . .
5        private PropertyChangeSupport changes
6           = new PropertyChangeSupport(this);
7
8        public void setMajor(String major)
9        {
10           String priorMajor = this.major;
11           this.major = major;
12           changes.firePropertyChange("major",
13               priorMajor, major);
14       }
15       public String getMajor()
16       {
17           return this.major;
18       }
```

Here we show a Student JavaBean that provides a bound property named "major". Whenever the major changes, the bean fires the PropertyChange event.

Note that the code is fairly trivial since the java.beans.PropertyChangeSupport class does most of the work!

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Implementing a Bound Property, cont'd

```
19        public void addPropertyChangeListener(
20           PropertyChangeListener l)
21        {
22            changes.addPropertyChangeListener(l);
23        }
24
25        public void removePropertyChangeListener(
26           PropertyChangeListener l)
27        {
28            changes.removePropertyChangeListener(l);
29        }
30    . . .
31    }
```


Completing the implementation of the bound property, here we show the methods that allow listeners to register and de-register for the property-change event.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Listening to a Bound Property

```
1    public class RespondToMajorChange
2        implements PropertyChangeListener
3    {
4        public void propertyChange(PropertyChangeEvent evt)
5        {
6            System.out.println("Property name: "
7                    + evt.getPropertyName());
8            System.out.println("Old value: "
9                    + evt.getOldValue());
10           System.out.println("New value: "
11                   + evt.getNewValue());
12       }
13       public static void main(String[] args)
14       {
15           RespondToMajorChange me =
16              new RespondToMajorChange();
17           Student s = new Student();
18           s.addPropertyChangeListener(me);
19           s.setMajor("Computer Science");
20       }
21   }
```

Here we show a class that implements the PropertyChangeListener interface, registers and object and then simply prints the property name, old value and new value.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Advanced JavaBean Techniques

- **Constrained** properties are like bound properties, but any listener can veto the change
- You can override or supplement the standard reflection and introspection of a JavaBean by providing a BeanInfo class
- Providing customized GUIs and property editors for tools that manipulate JavaBeans
- Providing a customized, serialized instance of a JavaBean

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Review Questions

- What differentiates a JavaBean from a normal Java class?
- What differentiates a JavaBean from an Enterprise JavaBean?
- What are the three kinds of features that a JavaBean can expose?

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter you learned:

- Why JavaBeans are useful and sample applications of JavaBeans
- About the three JavaBean features: events, methods and properties

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JEE Review

- Java Enterprise Edition
- Web applications

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Java Enterprise Edition?

- JEE is a set of technologies for server-side Java programming. It includes:
  - Servlets
  - JavaServer Pages
  - JavaServer Faces
  - Enterprise JavaBeans
  - Java Persistence API
  - plus much more
- There are many vendors implementing all or part of JEE

A primary advantage of JEE is that it's portable, both across JEE vendors as well as across platforms.

Note that JEE was formerly known as J2EE.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JEE Application Servers

- There are several JEE implementations available, including:
    - GlassFish
    - JBoss
    - WebSphere
    - WebLogic
    - Geronimo
    - Tomcat*

Tomcat is not considered to be a full JEE application server since it doesn't support EJB applications.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

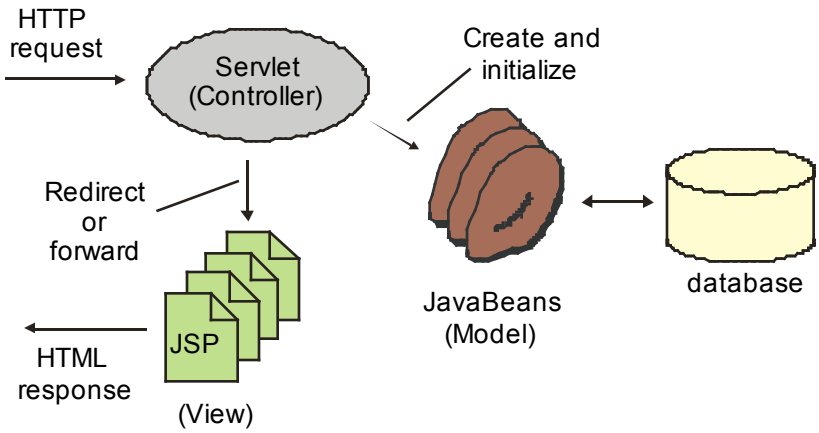
JEE Applications

- JEE supports two basic types of applications:
    - Web applications
    - EJB applications
- An application can include both!

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Model-View-Controller Architecture

- One very useful pattern you can use in JEE Web applications is referred to as MVC



To reduce code bulk in your JSPs, you can separate out the business logic and data into external JavaBeans. This will map nicely to the MVC architecture that we will cover in detail later.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Bean Scopes

- The Web container provides the following **scopes** so you can store bean references and provide differing visibility to the references
- Scope values, from least global to most global:
    - page
    - request
    - session
    - application

Instantiation is the process of creating an object and retrieving a reference to the new object. In a web application, you can store the reference in some of the predefined implicit objects covered earlier in the course. By choosing the scope (i.e. which implicit object), you can control how widely the reference is available.

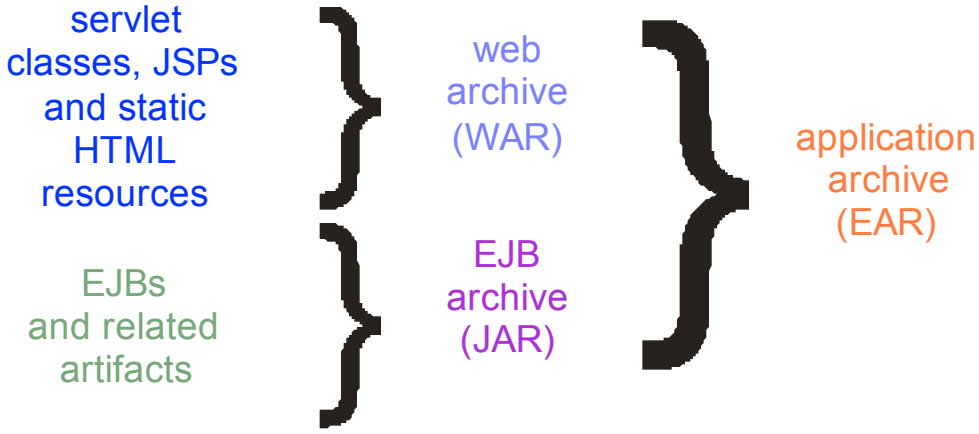
For example, if you store the reference at "page" scope, only the JSP that created the Bean can access it, but if you store the reference at "application" scope, any JSP or servlet that's in the same web application could retrieve the reference and access the Bean.

In a model-view-controller (MVC) web application, the controller often stores Bean references at request or session scope so that the JSPs that comprise the view can display them.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JEE Packaging Model

- Since JEE is a Java technology, programmers write JEE Java code and compile it in the normal fashion
- JEE provides various ways to package complete applications to make them easier to deploy



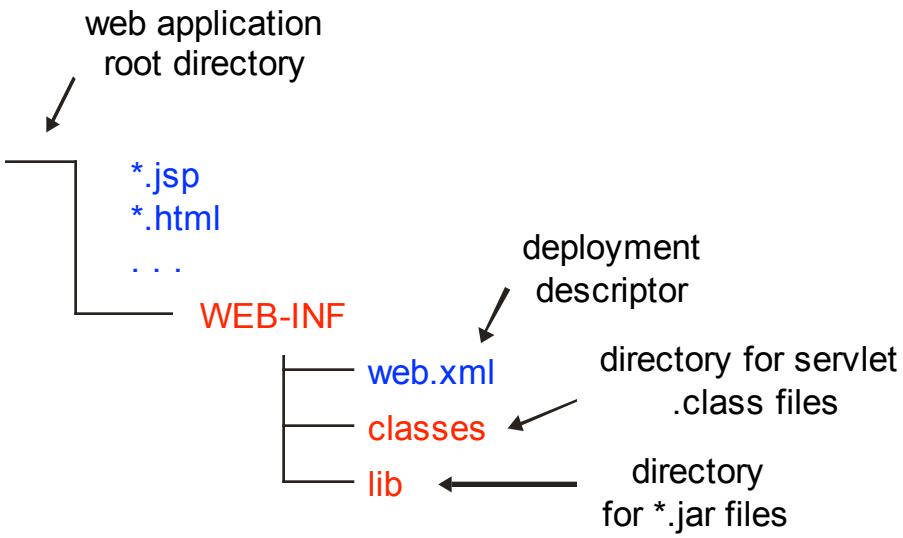
These archives are internally similar to ZIP files.

Note that each type of archive has its own deployment descriptor. The deployment descriptor shown on the last page was for a WAR file.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Web Application Directory Structure

- Web applications reside in a hierarchy of directories - the structure is defined by the Servlet specification



4 - 8

This figure shows the directory structure required for web applications. You create a root directory, and in it, place files and other sub-directories. The sub-directories must have the names shown here. The names are case-sensitive.

The root directory is where you deploy your JavaServer pages and static HTML files. You deploy your servlets into WEB-INF/classes directory or the WEB-INF\lib directory if you have packaged the servlets into JAR files.

The required "web.xml" file is known as the deployment descriptor -- it configures the web application. We will cover its syntax later.

While we show this as an explicit directory structure, it's also possible to package the entire thing into a single archive known as a .war file.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Web Deployment Descriptor

- The Servlet specification requires you to create a file named *web.xml* referred to as the deployment descriptor and store it in the WEB-INF directory
- The deployment descriptor describes the web application and is processed by the servlet container's deployment facility

```
<?xml version="1.0"?>
<web-app>
 <display-name>MyApp</display-name>
 <description>My Web Application</description>
 <welcome-file-list>
  <welcome-file>index.html</welcome-file>
 </welcome-file-list>
</web-app>
```

The deployment descriptor uses XML syntax to configure the web application. Like HTML, XML is text marked up with tags -- the start and end-tags delimit elements</i> that are processed by the container.

This page shows a deployment descriptor for a simple web application. It defines the standard XML declaration in line 1, followed by the required root element of "web-app". The "display-name" and "description" elements provide information about the web application. The "welcome-file-list" element lets you configure which files will be shown if the referencing URL does not include a file name. Using the web app shown on the last page, if the user enters a URL of http://localhost/myapp, the container will show the index.html file from the web application's root directory.

There are other elements possible in the deployment descriptor -- we will see more of them on the next few pages.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Deployment Descriptor Elements

- Servlet Initialization Parameters
- Session Configuration
- Servlet / JSP Definitions
- Servlet / JSP Mappings
- Mime Type Mappings
- Welcome File List
- Error Pages
- Security Constraints

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Forwarding versus Redirecting

- JEE components can transfer control to other components using **forwarding** or **redirecting**
- Forwarding is faster since it comprises a single request plus it allows the source and destination to share beans at **request** scope
- Redirecting is slower (two requests) and requires use of **session** scope, for sharing, but prevents problems if the user refreshes their browser

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The basics of writing servlets and JSPs
- The basics of tag libraries

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Architecture

- JSF View Components
- Managed Beans
- Navigation

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Requirements

- A Faces application needs:
  - Configuration of the Faces servlet in the Web deployment descriptor
  - A Faces configuration file
  - The JSF JARs in the **WEB-INF/lib** folder or provided by the application server
- The application also typically contains XHTML files or JSPs and **managed beans**

JEE application servers at version 5 or later include the required JSF libraries so that applications need not configure them in the Web app.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring the Faces Servlet

- In JSF, the Faces servlet acts as a **front controller** and must process **ALL** requests
- You configure the Faces servlet in the standard Web deployment descriptor

web.xml

```
1    <servlet>
2      <servlet-name>Faces Servlet</servlet-name>
3      <servlet-class>
4        javax.faces.webapp.FacesServlet
5      </servlet-class>
6    </servlet>
7    <servlet-mapping>
8      <servlet-name>Faces Servlet</servlet-name>
9      <url-pattern>*.faces</url-pattern>
10   </servlet-mapping>
```

You must configure the JSF servlet in web.xml and assign it a URL pattern. There is no requirement for the URL pattern, but the *.faces extension mapping shown here is common.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <?xml version="1.0" encoding="UTF-8"?>
2    <web-app id="WebApp_ID" version="3.0"
3        xmlns="http://java.sun.com/xml/ns/javaee"
4        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6          http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
7        <context-param>
8            <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
9            <param-value>.xhtml</param-value>
10       </context-param>
11       <context-param>
12           <param-name>javax.faces.PROJECT_STAGE</param-name>
13           <param-value>Development</param-value>
14       </context-param>
15       <servlet>
16           <servlet-name>Faces Servlet</servlet-name>
17           <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
18       </servlet>
19       <servlet-mapping>
20           <servlet-name>Faces Servlet</servlet-name>
21           <url-pattern>*.faces</url-pattern>
22       </servlet-mapping>
23   </web-app>
```

Configuring the Faces Servlet, cont'd

- Web.xml can also specify the **default extension** for view files, e.g. *xhtml* or *jsp*
- Web.xml can also specify the **project stage**, which controls the amount of diagnostics generated: Development, UnitTest, SystemTest, and Production

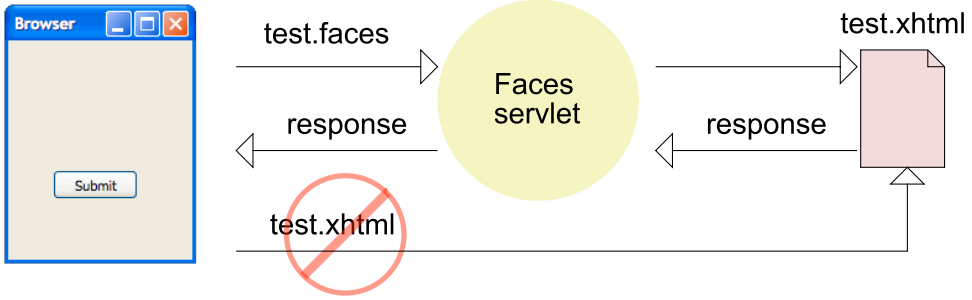
```
1    <context-param>
2      <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
3      <param-value>.xhtml</param-value>
4    </context-param>
5    <context-param>
6      <param-name>javax.faces.PROJECT_STAGE</param-name>
7      <param-value>Development</param-value>
8    </context-param>
```

If you don't specify a default extension, JSF uses .xhtml.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF URLs

- URLs from the client must follow the mapping that you configure in the Web deployment descriptor for the Faces servlet
- In this case, the Faces servlet strips the ".faces" from the URL and substitutes ".xhtml" to find a JSF page with that name



For JSF to work properly, all requests from the client must be routed through the Faces servlet. The servlet will use the input URL to find the JSF page for the request.

In this case, given the servlet configuration on the last page, the Faces servlet will process "test.xhtml" when the browser submits "test.faces".

There's nothing special about "*.faces" either -- you can use a different extension or so-called "path mapping" instead.

Also note that it's important that clients do NOT directly access a JSF page, since it won't work properly if they do.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Optional: Preventing Direct Page Access

- Faces pages must be processed by the Faces servlet and should not be accessed directly from a browser
- To disallow direct access, you can write a **security-constraint** element in the Web deployment descriptor

web.xml

```
1    <security-constraint>
2      <web-resource-collection>
3        <web-resource-name>Hide JSPs</web-resource-name>
4          <url-pattern>/index.xhtml</url-pattern>
5          <url-pattern>/addstudent.xhtml</url-pattern>
6           . . .
7        </web-resource-collection>
8        <auth-constraint>
9          <description>
10             Since we define no roles, no direct access
11             from the client
12         </description>
13       </auth-constraint>
14   </security-constraint>
```

This step is optional, but it's a good idea since it prevents clients from directly accessing JSF pagess and bypassing the Faces servlet. Note for this to really work, you need to write an entry for EVERY page in the Web application.

Also note that if the client does try to access a page directly with this in place, they client may get a security exception response that they might not understand.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Optional: Providing a Start Page

- JEE Web applications can define a **welcome-file** that acts as the start page of the application
- In JSF, the welcome JSP typically forwards to the first "real" page in the application, via the Faces servlet

web.xml

```
<welcome-file-list>
  <welcome-file>start.jsp</welcome-file>
</welcome-file-list>
```

start.jsp

```
<jsp:forward page="/index.faces"/>
```


This is mainly for backward compatibility with older application servers, especially Tomcat. On newer implementations, the "welcome-file" can directly specify index.faces or whatever your start page is.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF View Pages

- JSF 2 supports two approaches for writing pages:
    - Using XHTML
    - Using JSPs
- JSP support is mainly for backward compatibility with older versions of JSF

Note that if you use XHTML, the view files don't have to have .xhtml file extensions (though that is common). You can specify any file extension using the javax.faces.DEFAULT_SUFFIX context-parameter in web.xml.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Namespaces

- JSF defines two **namespaces** you can use in XHTML files
- The **core** namespace is view-independent and lets you perform validation, data conversion and so forth
- The **HTML** namespace lets you define input forms - you should use this library instead of standard HTML tags

```
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:h="http://java.sun.com/jsf/html">
. . .
</html>
```


Using XHMTL for pages instead of JSPs is a standard feature of JSF 2 and is the recommended approach (instead of using JSPs for the view).

In prior versions, XHTML was supported using a third-party JSF extension known as Facelets.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF JSP Tag Libraries

- JSF defines two tag libraries you can use in JSPs
- The **core** library is view-independent and lets you preform validation, data conversion and so forth
- The **HTML** library lets you define input forms - you should use this library instead of standard HTML tags

```
<%@taglib
  uri="http://java.sun.com/jsf/core" prefix="f"%>

<%@taglib
  uri="http://java.sun.com/jsf/html" prefix="h"%>
```


Using JSPs is not recommended for new applications and is primarly for backward compatibility with older version of JSF.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample JSF Page: XHTML

```
1    <html xmlns="http://www.w3.org/1999/xhtml"
2       xmlns:f="http://java.sun.com/jsf/core"
3       xmlns:h="http://java.sun.com/jsf/html">
4      <h:head>
5        <title>Add Student</title>
6      </h:head>
7       <h:body>
8        <h:form>
9          <p>Student name:<h:inputText
10             value="#{student.name}"/></p>
11         <p>Student ID: <h:inputText
12             value="#{student.id}"/></p>
13         <p>GPA: <h:inputText value="#{student.gpa}"/></p>
14         <p><h:commandButton value="Submit"
15            action="#{student.addStudent}"/></p>
16       </h:form>
17      </h:body>
18   </html>
```


Note that when we use XHTML, it's best to use the h:head and h:body elements instead of their standard HTML equivalents.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sample JSF Page: JSP

```
1    <%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
2    <%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
3    <html>
4      <head>
5        <title>Add Student</title>
6      </head>
7      <f:view>
8       <body>
9        <h:form>
10         <p>Student name:<h:inputText
11             value="#{student.name}"/></p>
12         <p>Student ID: <h:inputText
13             value="#{student.id}"/></p>
14         <p>GPA: <h:inputText value="#{student.gpa}"/></p>
15         <p><h:commandButton value="Submit"
16            action="#{student.addStudent}"/></p>
17       </h:form>
18      </body>
19    </f:view>
20   </html>
```

Note that if we use JSPs, we must explicilty specify an f:view - that's not necessary for XHTML-based pages.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Managed Beans

- JSF applications use **managed beans** to represent business data and logic
- Managed beans expose **properties** via get/set methods
- You can configure managed beans with **annotations** or in the Faces configuration file

```
1    public class Student
2    {
3    . . .
4      public int getId()
5      {
6       . . .
7      }
8      public void setId(int i)
9      {
10      . . .
11     }
12   }
```

faces-config.xml

```
<managed-bean>
 <managed-bean-name>
   student
 </managed-bean-name>
 <managed-bean-class>
   univ.Student
 </managed-bean-class>
 <managed-bean-scope>
  request
 </managed-bean-scope>
</managed-bean>
```

You can use the @ManagedBean and @SessionScope (or other scope) annotations to configure your managed beans - that way, you don't need to configure them in faces-config.xml. The annotations are a new feature in JSF 2.

Managed beans expose properties by defining get/set methods and can provide logic by providing methods.

A JSF page can access a managed bean's properties and methods using the JSF Expression Lanaguage.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to the JSF Expression Language

- JSF provides an **expression language** so that page authors can access managed bean properties as well as several predefined objects (e.g. **cookie**)
- JSF version 1.2 supports the **Unified Expression Language**, which unifies the JSF EL with the standard JSP expression language
- JSF expressions reside within JSF tags and have basic syntax **#{expr}**

```
#{student.id}

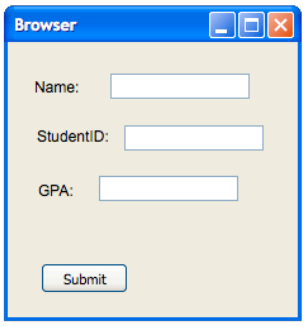
#{student.gpa}
```


When you write a JSF expression that references a property, behind the scenes, JSF calls the bean's get/set methods.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JSF Components

- A JSF page comprises one or more **components**, each of which displays content and/or accepts user input
- The Faces servlet uses JSF tags in the JSF page to build a **component tree** of objects on the server
- The standard JSF components generate HTML

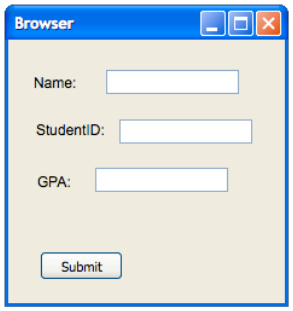
JSF Tag		Component class
<h:form>		UIForm
<h:inputText>		UIInput
<h:inputText>		UIInput
<h:inputText>		UIInput
<h:commandButton>		UICommand

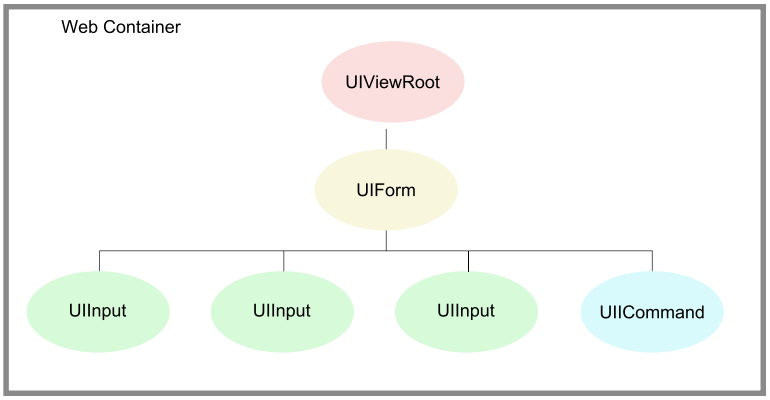
In a JSF JSP, you use JSF custom actions (tags) to define the form. At runtime, the JSF controller builds a component tree from the custom actions, using component classes defined by JSF.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Component Tree

- When the Faces servlet processes a request, it first builds a **component tree** and lets the tree participate in processing the request and generating a response

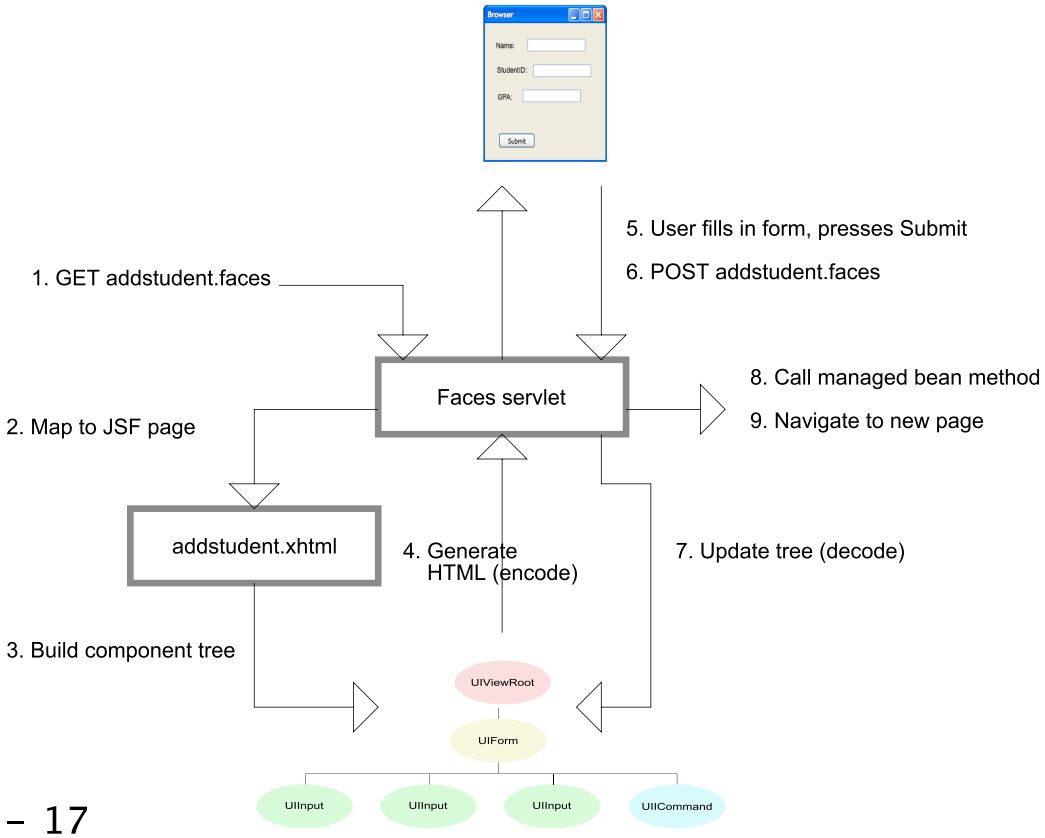




Components are the heart of a JSF application -- they provide the user interface. JSF borrowed this notion from client-side GUIs like Visual Basic or Java Swing.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Typical Request Processing



To start this flow, the user first enters the URL for a page, which typically has an extension of .faces. This invokes the Faces servlet, which finds the matching JSF page and processes the page.

The JSF tags in the page collaborate to build the component tree. The Faces servlet then renders the page and component tree. All non-JSF text (e.g. HTML paragraphs) are passed back verbatim, and the JSF components render their HTML equivalent. For example, the h:inputText tag's component generates an HTML tag like <input type="text">. This process is known as encoding.

The user then fills in the form and submits it back to the same URL. The Faces servlet finds the component tree and lets the tree work with the user input, a process known as decoding. The result is that the each component captures its tag's user input and stores in the specified property in the managed bean.

The Faces servlet then invokes the method specified by the Submit button's JSF expression. The method processes the request (examining the user-input) and returns a string which the servlet uses to navigate to a new page.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Invoking an Application Method

- When the user submits the form, the Faces servlet lets the component tree **decode** it and update managed-bean properties, then invokes the method specified on the submit button

```
<h:commandButton
   value="Add student"
   action="#{student.addStudent}"/>
```

```
private int id;
private double gpa;
private String name;
. . .

public String addStudent()
{
   // process id,gpa,name as required
   // store objects at request or session scope

   if (ok) return "success"
   else return "bad-news";
}
```


Before calling "addStudent", the framework has initialized the "id", "gpa" and "name" properties in the managed bean, so "addStudent" can use those values to perform data processing.

Based on the results of processing, the "addStudent" method returns a mapping string -- the Faces servlet uses this string to determine to which page to navigate.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Accessing the Servlet Environment

- The processing method in the managed bean typically accepts no parameters
- If the method needs to access the JSF and/or servlet environment (e.g. the servlet request), it can retrieve the JSF **external context**

```
1    ExternalContext context =
2     FacesContext.getCurrentInstance().getExternalContext();
3
4    HttpServletRequest request =
5      (HttpServletRequest)context.getRequest();
6
7    Cookie[] cookies = request.getCookies();
```


Often when processing a request, the method needs to work with the servlet or JSF environment. To gain access, the method can retrieve an external context and use it to retrieve objects.

For example, here we show using the ServletRequest object to retrieve the array of HTTP cookies passed on a request.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Navigation

- You can configure **navigation rules** in the Faces configuration file
- The processing method returns a logical string that Faces uses to navigate to another page

```
1    <navigation-rule>
2      <from-view-id>/addstudent.xhtml</from-view-id>
3      <navigation-case>
4        <from-outcome>success</from-outcome>
5        <to-view-id>/confirmation.xhtml</to-view-id>
6      </navigation-case>
7      <navigation-case>
8        <from-outcome>bad-news</from-outcome>
9        <to-view-id>/failure.xhtml</to-view-id>
10     </navigation-case>
11   </navigation-rule>
```


Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How to configure JSF
- How JSF uses URLs
- The fundamental JSF processing model

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Request Processing and Navigation

- JSF Request Lifecycle
- Navigation

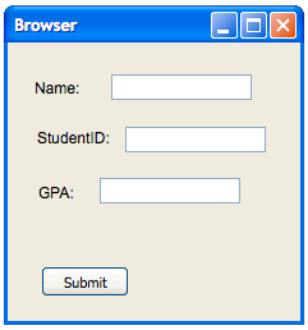
Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Components

- A JSF page comprises one or more **components**, each of which displays content and/or accepts user input
- The Faces servlet uses JSF tags in the JSF page to build a **component tree** of objects on the server
- The standard JSF components generate HTML

JSF Tag	Component class
<h:form>	UIForm
<h:inputText>	UIInput
<h:inputText>	UIInput
<h:inputText>	UIInput
<h:commandButton>	UICommand

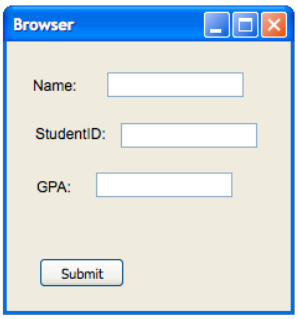


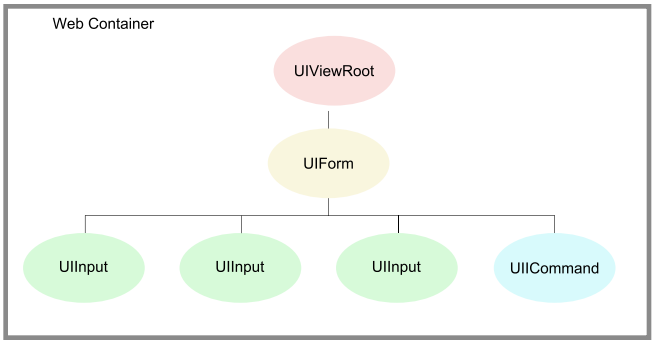
In a JSF page, you use JSF custom actions (tags) to define the form. At runtime, the JSF controller builds a component tree from the custom actions, using component classes defined by JSF.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Component Tree

- When the Faces servlet processes a request, it first builds a **component tree** and lets the tree participate in processing the request and generating a response



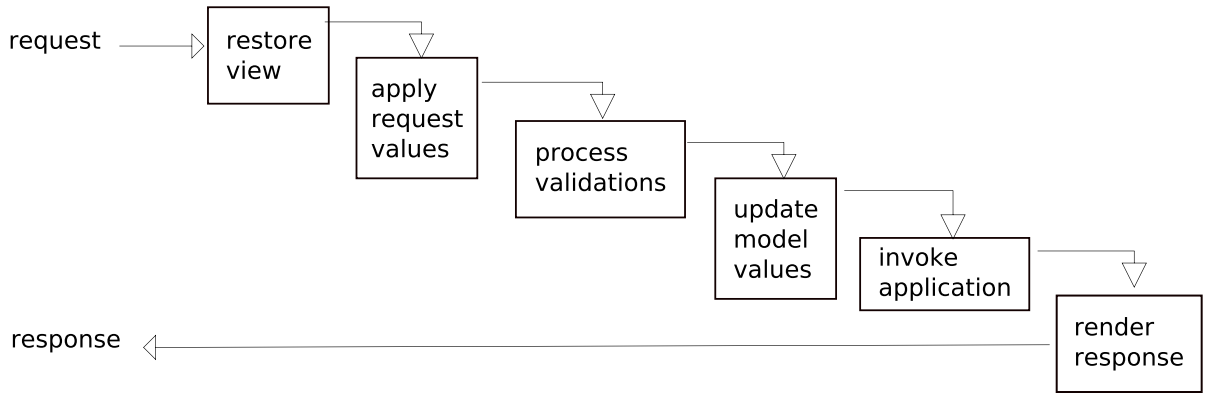


Components are the heart of a JSF application -- they provide the user interface. JSF borrowed this notion from client-side GUIs like Visual Basic or Java Swing.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Request/Response Lifecycle

- This figure shows the phases of a typical postback request



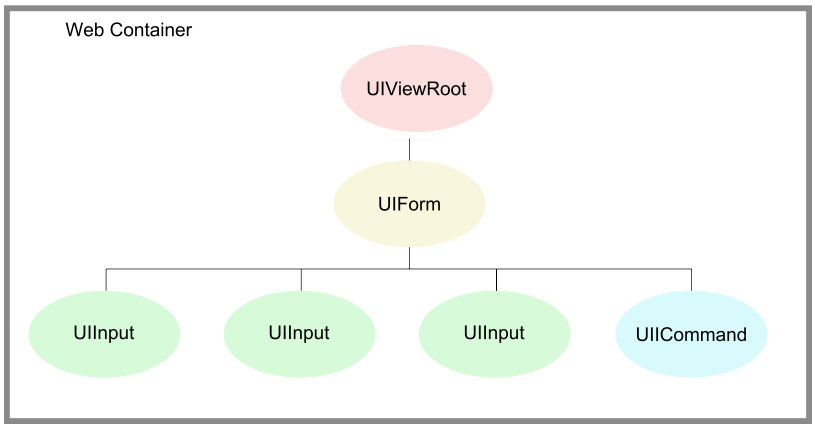
This figure is a simplification of the entire request lifecycle, but it shows enough detail to get started with.

Note that the figure assumes that it's a POSTed request: on the original GET, the only phases that run are "Restore View" (to build the component tree) and "Render Response" to generate the initial page.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Restore View Phase

- In this phase, JSF builds or restores the component tree



If this is the first, GET request for a page, JSF builds the component tree. If it's a POSTback (or if the component tree still exists in memory from a previous request), then JSF restores the component tree as necessary.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

View State

- JSF must rebuild or restore the view state (component tree) when the postback occurs
- JSF supports two ways to save this state:
    - On the server as an **HTTP session** variable
    - On the client via hidden form field(s)
- You can specify the value as a **context-param** in the Web deployment descriptor

```
1    <web-app . . .>
2     . . .
3      <context-param>
4        <param-name>
5         javax.faces.STATE_SAVING_METHOD
6        </param-name>
7        <param-value>client</param-value>
8      </context-param>
9     . . .
10   </web-app>
```

In either case, the view is tagged with a unique ID.

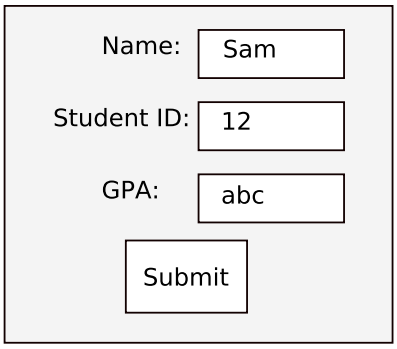
If you use "server", then the component tree is maintained on the server in an HTTP session. When the postback occurs, JSF can re-associate the component tree with the request, typically matching the view ID to the session-scoped variable.

If you use "client", then the component tree is "serialized" and sent to the client as the contents of one or more hidden form fields. Upon postback, JSF de-serializes the hidden form field's contents to recreate the component tree. Note that the exact mechanism to store the state is implementation dependent, but most will encrypt the state before putting it in the hidden form field.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Apply Request Values Phase

- Requests typically contain **parameters** for example, an input box in a form
- Parameters typically correspond to a **property** in a managed bean
- During this phase, the component tree processes the request parameters and converts them to the property type




```
public class Student
{
  . . .
  public double getGpa() {}
  public void setGpa(double gpa) {}
  . . .
}
```


If the conversion fails, JSF redisplays the input page.

Before redisplaying the input page, JSF first creates and queues an error-message string - the input page can use the h:message or h:messages tags to display them.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

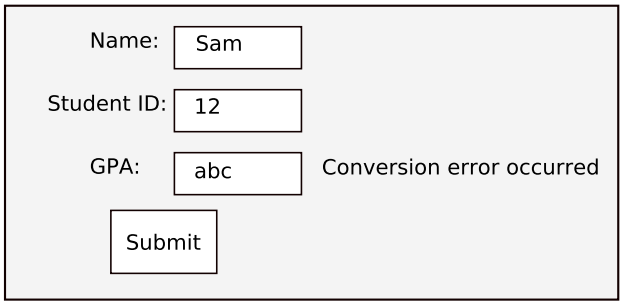
Displaying Conversion Errors

- If a conversion error occurs, the component stores an error message in the FacesContext
- You can use the **h:message** tag to display conversion errors if the page is redisplayed
- We will cover more about conversion errors later

student.xhtml

```
<p>
  GPA:
  <h:inputText id="gpa" value="#{student.gpa}" />
  <h:message for="gpa"></h:message>
</p>
```

After conversion error and redisplay



The h:message tag requires you to tag the input field with a unique ID so that h:message can reference it.

There is also an h:messages tag that lets you display all of the error messages in a list.

Later, we will cover how to define your own error message strings, but here we use the default (built-in) message.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

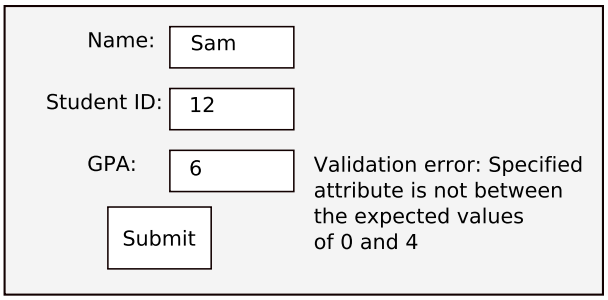
Validation Phase

- During this phase, the component tree applies validation rules to the data supplied in the *apply request values* phase
- If validation fails, JSF redisplays the input page, which can display the validation error(s)
- JSF includes several standard validators and you can code your own
- We will cover more about validation later

student.xhtml

```
<p>
 GPA:
 <h:inputText id="gpa" value="#{student.gpa}" >
   <f:validateDoubleRange minimum="0.0"
      maximum="4.0"/>
   </h:inputText>
 <h:message for="gpa"></h:message>
</p>
```

After validation error and redisplay



JSF provides several built-in validators like validateDoubleRange. In addition, you can create your own.

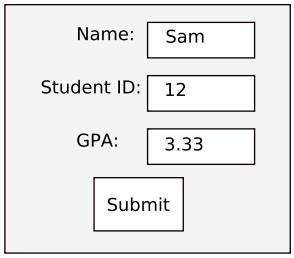
Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Update Model Phase

- During this phase, the component tree calls the "set" methods on managed beans to assign the potentially converted and validated request values

student.xhtml

```
<p>
 Name:
 <h:inputText id="name" value="#{student.name}" />
</p>
<p>
 Student ID:
 <h:inputText id="studentID"
  value="#{student.studentID}" />
</p>
<p>
 GPA:
 <h:inputText id="gpa" value="#{student.gpa}" />
</p>
```




```
public class Student
{
  . . .
  public String getName() {}
  public void setName(String name) {}
  public int getStudentID() {}
  public void setStudentID(int id) {}
  public double getGpa() {}
  public void setGpa(double gpa) {}
  . . .
}
```


Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Invoke Application Phase

- During this phase, JSF invokes the "action" method in a managed bean
- You specify the action method in the tag that allows the user to submit the page (e.g. an **h:commandButton**
- The action method typically returns a String that JSF processes against the navigation rules defined in faces-config.xml

student.xhtml

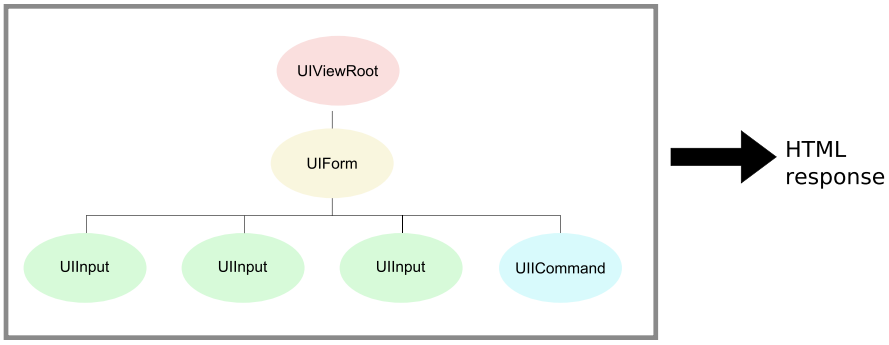
```
<p>
  <h:commandButton value="Submit"
     action="#{student.updateStudent}" />
</p>
```

```
public class Student
{
  . . .
  public String updateStudent()
  {
     // update Student in DB
     if(no_problem)
       return "success";
     else
       return "database-error";
   }
}
```


Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Renderer Response Phase

- In this phase, the component tree generates the actual response, typically in HTML
- This generation is sometimes referred to as **encoding** the response
- Most components delegate the actual output generation to a **renderer**



This phase also happens on the original GET request to the page. That's where the component tree encodes the original HTML that contains the input form.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Backing Beans

- Sometimes it's convenient to have a references to the UI component objects
- A class that maintains such references is referred to as a **backing bean**
- You bind references to backing bean properties by writing JSF EL

student.xhtml

```
<h:inputText id="gpa" value="#{student.gpa}"
        binding="#{student.inputNameComponent}">
```

```
public class Student
{
   . . .
   private UIInput inputNameComponent;

   public UIInput getInputNameComponent()
   {
      return inputNameComponent;
   }

   public void setInputNameComponent(UIInput inputNameComponent)
   {
      this.inputNameComponent = inputNameComponent;
   }
   . . .
   public String updateStudent()
   {
      System.out.println("input component ID: " +
           inputNameComponent.getId());
      . . .
   }
}
```


Some JSF visual design tools generate such backing beans as a matter of course.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Navigation

- Navigation is processing of moving from one page to the next
- JSF lets you configure **navigation rules** in the faces-config.xml file

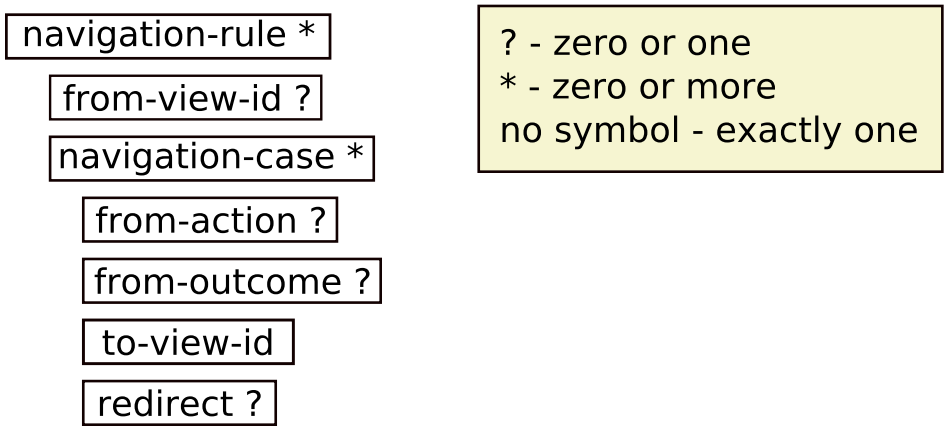
```
1    <faces-config>
2      <navigation-rule>
3        <from-view-id>/student.xhtml</from-view-id>
4        <navigation-case>
5          <from-outcome>success</from-outcome>
6          <to-view-id>/student-updated.xhtml</to-view-id>
7        </navigation-case>
8        <navigation-case>
9          <from-outcome>database-error</from-outcome>
10         <to-view-id>/badDB.xhtml</to-view-id>
11       </navigation-case>
12     </navigation-rule>
13   . . .
14   </faces-config>
```

Note that in the "view-id" elements, the leading slash is required. Also note that we specify actual JSPs here; not the URL that comes from the client.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Navigation Rule Syntax

- The XML syntax for navigation has several optional and required elements



The "from-action" element is rarely used.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Static Navigation

- With **static navigation**, navigation doesn't depend on anything except the navigation rule
- In other words, the navigation always works the same
- Note that if no navigation rule matches a given action, JSF redisplays the current page

test.xhtml

```
<h:commandButton value="Get help"
   action="help"/>
```

faces-config.xml

```
<navigation-rule>
  <from-view-id>/test.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>help</from-outcome>
    <to-view-id>/help.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```


Note that the "action" attribute's value in the JSP is a simple string, not a JSF EL expression.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Dynamic Navigation

- With **dynamic navigation**, JSF calls an **action** method and applies a navigation rule based on its String return value
- If the method returns **null**, JSF redisplays the current page

test.xhtml

```
<h:commandButton value="Submit"
   action="#{hello.mymethod}" />
```

faces-config.xml

```
<navigation-rule>
  <from-view-id>/test.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/hello.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>bad-name</from-outcome>
    <to-view-id>/badname.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```


Again, if there's no matching navigation rule, then JSF redisplays the current page. For example, in this case, if mymethod returns "hello", there's no matching "from-outcome", so JSF will redisplay test.jsp.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Redirection

- Normally, JSF uses a *forward* to navigate, but you can specify the **redirect** element so that JSF uses redirection instead

test.xhtml

```
<h:commandButton value="Submit"
   action="#{hello.mymethod}" />
```

faces-config.xml

```
<navigation-rule>
  <from-view-id>/test.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/hello.xhtml</to-view-id>
     <redirect />
  </navigation-case>
  <navigation-case>
    <from-outcome>bad-name</from-outcome>
    <to-view-id>/badname.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```


Redirection is slower than forwarding, but causes the browser to start a new request, thus perhaps avoiding issues due to the user refreshing the browser.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Navigation Wildcards

- You can use the * character at the end of a *from-view-id* so that the rule applies to multiple input pages
- Note that if you omit *from-view-id*, the rule applies to **all** input pages

test1.xhtml

```
<h:commandButton value="Help"
   action="help" />
```

test2.xhtml

```
<h:commandButton value="Help"
   action="help" />
```

faces-config.xml

```
<navigation-rule>
 <from-view-id>*</from-view-id>
 <navigation-case>
  <from-outcome>help</from-outcome>
  <to-view-id>/help.xhtml</to-view-id>
   <redirect />
 </navigation-case>
```


In this example, the wildcard specified has the same effect as omitting the "from-view-id" since it applies to all pages.

But you're not limited to this - you could write a wildcard that looks like:

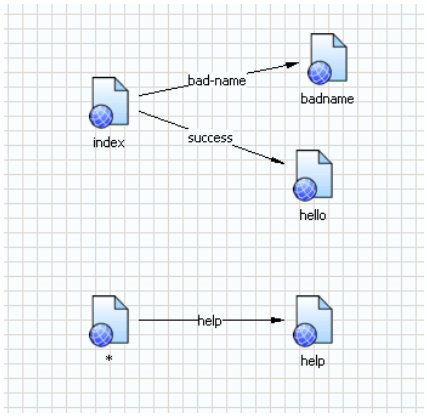
```
<from-view-id>/mydir/*</from-view-id>
```

to only work on JSPs from the "mydir" directory.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a GUI Tool for Navigation Rules

- Many IDEs include a graphical tool to help you generate navigation rules without having to write the raw XML



Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Additional JSF2 Navigation Features

- JSF Version 2 added several navigation capabilities:
    - Implicit navigation
    - Conditional navigation

One focus of JSF2 is to simplify the programming model.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Implicit Navigation

- Implicit navigation simplifies navigation by eliminating the need to write XML navigation rules
- Implicit navigation is useful for both **static** and **dynamic** navigation

```
1    <!-- static, but no navigation rule needed -->
2    <h:commandButton action="page2.xhtml" value="Submit" />
```

```
1    // dynamic, but no navigation rules needed
2    public String myActionMethod()
3    {
4      if(someCondition == true)
5        return "page1.xhtml";
6      else
7        return "page2.xhtml";
8    }
```

Note that you don't even need to use the ".xhtml" suffix - JSF2 will automatically append it. So the first example could be even simpler:

```
<h:commandButton action="page2" value="Submit" />
```

and the second:

```
public String myActionMethod()
{
  if(someCondition == true)
    return "page1";
  else
    return "page2";
}
```

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Conditional Navigation

- Conditional navigation lets you define conditions for navigation cases in XML

```
1    <navigation-rule>
2       <from-view-id>/addStudent.xhtml</from-view-id>
3       <navigation-case>
4          <from-outcome>success</from-outcome>
5          <if>#{bean.gpa &gt; 3.0}</if>
6          <to-view-id>/honorRoll.xhtml</to-view-id>
7       </navigation-case>
8        <navigation-case>
9          <from-outcome>success</from-outcome>
10         <to-view-id>/normal.xhtml</to-view-id>
11      </navigation-case>
12   </navigation-rule>
```


Essentially, this migrates navigation logic from the action method (written in Java) to the faces-config.xml.

Note that JSF evaluates the rules from top to bottom, so the topmost one takes precedence in this case.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About the JSF request-processing lifecycle
- How to write navigation rules

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Managed Beans and the JSF EL

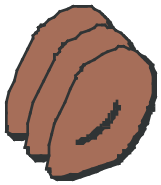
- What is a Managed Bean?
- Wiring
- The JSF Expression Language

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Managed Bean?

- A **managed bean** class is a Java class that adheres to parts of the JavaBean specification:
    - Must have a public zero-argument constructor
    - Can expose **properties** via get/set methods
- JSF handles creating and optionally initializing managed beans for the application
- You configure managed beans either with **annotations** or in the faces-config.xml file



If you application will store managed beans at session scope, then the bean class should implement the java.io.Serializable interface.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Managed Beans

- JSF lets you write **expression language** expressions that bind to properties in a managed bean
- JSF provides a **dependency injection** facility that lets you initialize and *wire* beans together

Student.java

```
@ManagedBean(name="student2")
@SessionScope
public class Student implements Serializable
{
 . . .
}
```

student.xhtml

```
<p>
  Student ID:
  <h:inputText value="#{student2.studentID}" />
</p>
```

JSF expression language expressions are delimited by #{expr}.

JSF will defer creating a bean until it's referenced in a value-binding expression.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Bean Annotations

- In JSF version 1, you had to define managed beans in faces-config.xml, but JSF version 2 and later provide **annotations**
- JSF provides its own annotations from the **javax.faces.bean** package
- Java Enterprise Edition (JEE) version 6 provides similar, overlapping annotations that are not limited to JSF-based applications from the **javax.enterprise.context** package

The JEE annotations are from the JSR 299: Contexts and Dependency Injection (CDI) specification. The equivalent annotation to the JSF @ManagedBean is @Named.

If you are using a JEE Version 6 application server, then you can stick to the CDI annotations, but if you are using an older application server, you can use the JSF annotations.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Bean Scopes

- A **scope** is a table of name/value pairs
- JSF supports the following scopes:
    - none
    - request
    - view
    - session
    - application
- Request, session and application scope are the same as in non-JSF Web applications
- The **none** scope is useful for beans that are used temporarily (e.g. to perform a calculation) and don't need to be referenced later

---

When you configure a managed bean, you are required to specify one of these four scopes.

Note that "application" scope should only be used for read-only data, otherwise the application will not be thread safe.

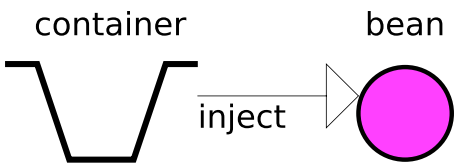
View scope is particularly useful for Ajax applications.

Note that JSF2 also supports conversation scope and custom scopes. We don't cover those here.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Dependency Injection and Wiring

- **Dependency injection** is a technique in which a runtime (container) provides values to beans
- DI was made famous by the Spring Framework, but has been widely adopted elsewhere
- JSF provides a simple DI facility to initialize beans with simple values and to **wire** beans together by injecting references





Dependency injection is a form of Inversion of Control</i> where the container supplies a bean with its dependencies.

The term "Dependency injection" was coined by Martin Fowler.

JSF only supports "setter" injection, which means that JSF provides values by calling a JavaBean-style "set" method. Spring also supports "constructor" injection, in which the Spring container initializes beans as it creates them.

If you need heavy-duty injection, you can integrate Spring with JSF and use Spring to define and initialize (inject) managed beans.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Injecting Simple Values

- You can initialize simple managed bean properties using dependency injection

```
1    @ManagedBean(name = "student2")
2    @SessionScoped
3    public class Student implements Serializable
4    {
5      private int studentID;
6      private String name;
7
8      @ManagedProperty(value="3.33")
9      private double gpa;
10   . . .
11   }
```

When you inject a simple property, JSF will attempt to convert the provided (string) value to the property's actual type.

JSF provides default conversions for primitives.

You can also inject in the faces-config.xml:

```
<managed-bean>
  <managed-bean-name>student2</managed-bean-name>
  <managed-bean-class>
     university.Student
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <managed-property>
    <property-name>gpa</property-name>
    <value>3.33</value>
  </managed-property>
</managed-bean>
```

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Injecting Lists

- For bean properties that implement the **java.util.List** interface, you can inject multiple values

```
1     <managed-property>
2       <property-name>majors</property-name>
3       <list-entries>
4         <value>History</value>
5         <value>Physics</value>
6       </list-entries>
7     </managed-property>
```

```
          1    public class Student
          2    {
          3      private List<String> majors;
          4
          5      public List<String> getMajors() {}
          6      public void setMajors(List<String> l) {}
          7      . . .
 7 - 8    8    }
```

JSF will create and populate the list based on the managed-property configuration.

Apparently, there's no way to do this with annotations.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Injecting Maps

- For bean properties that implement the **java.util.Map** interface, you can inject multiple name/value pairs

```
1     <managed-property>
2       <property-name>professors</property-name>
3       <map-entries>
4         <map-entry>
5           <key>John Smith</key>
6           <value>Chemistry 101</value>
7         </map-entry>
8         <map-entry>
9           <key>Sue Jones</key>
10          <value>The US Civil War</value>
11        </map-entry>
12      </map-entries>
13    </managed-property>
```

```
public class Student
{
  . . .
   private Map<String,String> professors;

   public Map<String, String> getProfessors() {}
   public void setProfessors(Map<String, String> professors) {}
}
```


JSF will create and populate the map based on the managed-property configuration.

Appararently, there's no way to do this with annotations.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Wiring Bean References

- Instead of injecting simple values or collections, you can inject a reference to another managed bean using a value binding expression

```
1    @ManagedBean(name = "advisor1")
2    @SessionScoped
3    public class Advisor implements Serializable
4    {
5      . . .
6    }
7
8    @ManagedBean(name = "student2")
9    @SessionScoped
10   public class Student implements Serializable
11   {
12     @ManagedProperty(value="#{advisor1}")
13     private Advisor theAdvisor;
14     . . .
15   }
```

```
public class Student
{
  . . .
   private Advisor advisor;

   public Advisor getAdvisor() {}
   public void setAdvisor(Advisor advisor) {}
}
```


You can also wire references in the faces-config.xml:

```
<managed-bean>
  <managed-bean-name>advisor1</managed-bean-name>
. . .
</managed-bean>

<managed-bean>
  <managed-bean-name>student2</managed-bean-name>
  <managed-bean-class>
     university.Student
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <managed-property>
    <property-name>advisor</property-name>
    <value>#{advisor1}</value>
  </managed-property>
  . . .
</managed-bean>
```

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Rules for Bean Scopes

- When you wire beans, you must ensure that the following scope rules are followed:

Source bean scope	Compatible scopes for referenced beans
none	none
request	none, request, session, application
session	none, session, application
application	none, application

The basic rule is that referenced beans must be at least as global as the source bean.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Expression Language

- All JSF EL expressions use the **#{expr}** syntax to differentiate them from the JSP EL, which uses **${expr}**
- You can use JSF expression language in three places:
    - JSF view pages (.xhtml or .jsp)
    - faces-config.xml
    - Within Java code, especially action methods

```
#{student2.updateStudent}
```


Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Why Another Expression Language?

- Because of the complex JSF request lifecycle, the JSP EL is not sufficient:
    - JSP EL evaluates only once during the lifecycle, while JSF EL needs to execute at multiple points in the lifecycle
    - JSP EL is read-only
    - JSP EL does not provide syntax to call methods
    - JSP EL doesn't work in XHTML files

In addition to the JSP EL and the JSF EL, there is a Unified Expression Language implemented by JSP 2.1 that combines the syntaxes.

With the Unified Expression Language, JSPs in JSF applications can use ${expr} syntax for "immediate evalutation" or #{expr} for "deferred evaluation". In other words, in JSP 2.1, JSTL tags like c:forEach can use the #{expr} syntax.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF EL Operators

- The JSF expression language defines the familiar four-function calculator math operators, relational and logical operators
- Note that many of the operators have more "human readable" synonymns

Category	Operators	Synonyms
Arithmetic	+,-,*,/,%	div,mod
Relational	==,!=,<,>,<=,>=	eq,ne,lt,gt,le,ge
Logical	&&,||,!	and,or,not
Member access	.,[]	
Conditional	? :	
Miscellaneous	empty	

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Operator Examples

Expression	Result
#{3 * 4}	12
#{5 gt 3}	true
#{(10 div 2) eq 8}	false
#{empty param.age}	false if "age" parameter is not null and has non-zero length
#{student2.gpa * 5}	The product of the "gpa" property of a student2 managed bean and the value 5

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF EL Implicit Variables

- To make it easy to accomplish common tasks, the EL defines implicit objects that you can reference from within an expression

Variable	Type	Description
param	Map	Returns a String for a form param
paramValues	Map	Returns String[ ] for form param
header	Map	Returns a String for an HTTP header
headerValues	Map	Returns String[ ] for HTTP header
facesContext	FacesContext	Provides access to JSF context values
cookie	Map	Returns a cookie value given a name
initParam	Map	Returns an init param given a name
requestScope	Map	Maps all request scope objects
sessionScope	Map	Maps all session scope objects
applicationScope	Map	Maps all app scope objects
view	UIViewRoot	Provides access all view components

The Map collection is like a phone book -- it's a keyed collection that stores values associated with a unique key. Thus the EL can use it to store/retrieve attributes from the various scopes (the attribute name is the key). To retrieve a value, you can use either the dot (.) or [ ] operators and specify the key.

The difference between param and paramValues is that the latter evaluates to an array of Strings for request parameters that have multiple values. The same applies for the header and headerValues variables.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Implicit Variable Examples

#{header.accept} = */*

#{header['accept']} = */*

#{header['user-agent']} = Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1;
Trident/4.0; .NET CLR 2.0.50727; .NET CLR 1.1.4322;
MS-RTC LM 8; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)

#{facesContext.responseComplete} = false

This page shows some expressions that use implicit objects and sample values from a request.

Note that in general, if an implicit variable is a Map, you can use either the . (dot) operator or square brackets to retrieve a value given a key. Note however that for "user-agent", we must use the square brackets and quotes or else the EL will interpret the hyphen as the subtraction operator. In other words, this expression:

```
#{header.user-agent}
```

returns the value zero!

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Evaluating Expressions in Java

- Sometimes, it's useful to evaluate EL expressions in Java code, such as action methods in a managed bean

```
1    FacesContext ctx = FacesContext.getCurrentInstance();
2    Application app = ctx.getApplication();
3
4    // JSF 1.1
5    ValueBinding binding =
6      app.createValueBinding("#{student2.gpa * 5}");
7    Double gpa = (Double)binding.getValue(ctx);
8
9    // JSF 1.2 and later
10   ExpressionFactory fact = app.getExpressionFactory();
11   ELContext elContext = ctx.getELContext();
12   ValueExpression valExpr =
13      fact.createValueExpression(elContext,
14      "#{student2.gpa * 5}", Double.class);
15   gpa = (Double)valExpr.getValue(elContext);
```


Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How to use dependency injection to initialize and wire managed beans
- About the syntax of the JSF expression language

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Components

- Component Architecture
- Output Components
- Input Components
- Selection Components

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF is a Component-Based Framework

- Unlike other JEE Web frameworks such as Struts, JSF is based on a component model
- JSF component-model programming is conceptually similar to client-side GUI programming using the Java Swing API:
    - Components present a user interface
    - Developers can optionally use visual design tools
    - Components generate events when the user interacts with them

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Provides a Pluggable Architecture

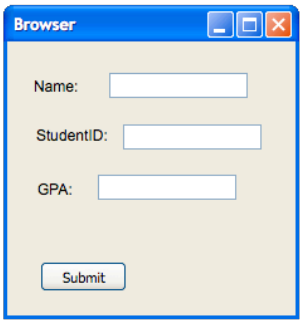
- JSF components often delegate some of their function to other code, for example:
    - Data conversion
    - Validation
    - Rendering
- This pluggable architecture lets developers customize JSF components by plugging in alternates, for example a custom renderer
- This architecture also conceivably lets JSF work with non-HTML environments

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

How Does JSF Create Component Objects?

- JSF applications typically use JSF tags to compose pages
- JSF invokes the tags during the JSF lifecycle and creates the **component tree**

JSF Tag		Component class
<h:form>		UIForm
<h:inputText>		UIInput
<h:inputText>		UIInput
<h:inputText>		UIInput
<h:commandButton>		UICommand

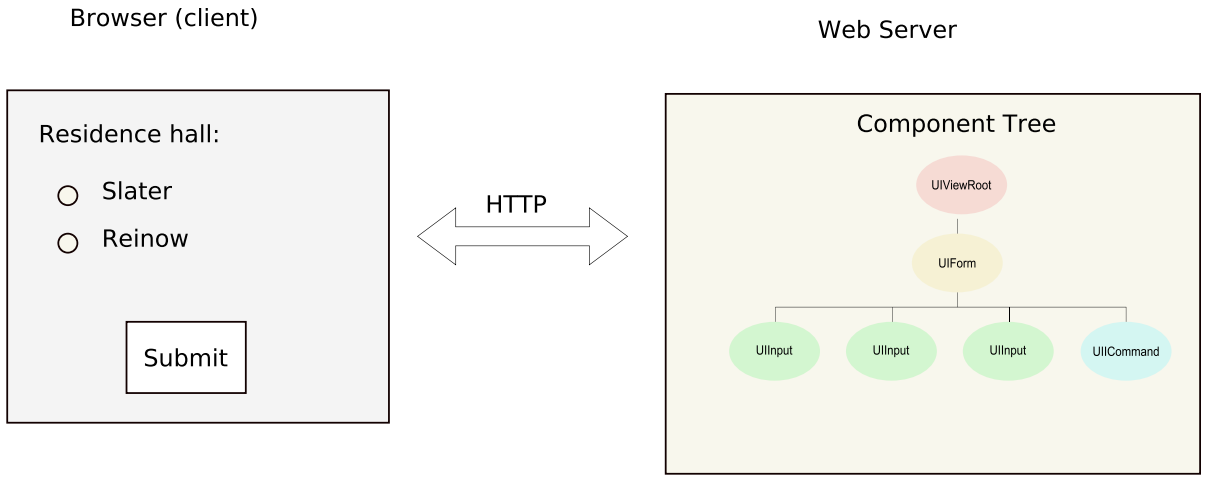


JSF applications are not required to use JSPs. Many JSF developers prefer to use an alternative view technology, for example Facelets, which uses XHTML to specify the components.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The Client-Server Model

- Unlike client-side GUIs, JSF is a client/server system, which has implications for responsiveness and event handling



Since JSF applications run as Web applications, they work differently than client-side applications. In particular, when the user interacts with a component, the interation occurs on the client, with no automatic notification to the component, which runs on the server.

For example, if the user clicks a radio button on a Web form, that event is not normally transmitted to the server until the user submits the entire page.

Later, we will discuss how to use client-side scripting to get more immediacy.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Tags

- JSF provides two commonly used sets of tags:
    - The **core** tags which are independent of the *render kit* (output generation)
    - The **html** tags which generate or operate on HTML and related Web technologies

```
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:h="http://java.sun.com/jsf/html">

...

</html>
```

If you are using JSPs instead of Facelets XHTML, you use JSP tag libraries instead of the XHTML namespaces:

```
<%@ taglib uri="http://java.sun.com/jsf/core"
  prefix="f" %>

<%@ taglib uri="http://java.sun.com/jsf/html"
   prefix="h" %>
```

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Core Tags

Tag	Summary
actionListener	Creates an action listener and associates it with a component
attribute	Associates an attribute with a component
convertDateTime	Associates a date/time converter with a component
converter	Creates a specified converter and associates it with a component
convertNumber	Associates a number converter with a component
facet	Adds a facet to a parent component (e.g. a DataTable)
loadBundle	Loads a resource bundle
param	Associates a name/value pair with a component
selectItem	Represents a choice in a list
selectItems	Represents a set of choices in a list
subView	Creates a sub-view container
validator	Creates a specfied validator and associates it with a component
validateDoubleRange	Creates a validator and associates it with a component
validateLength	Creates a validator and associates it with a component

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Core Tags, cont'd

Tag	Summary
validateLongRange	Creates a validator and associates it with a component
valueChangeListener	Creates value-change listener and associates it with a component
view	Creates a component that contains all other page components
verbatim	Outputs its body as verbatim text

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HTML Tags

Tag	Summary
column	Represents a column in a DataTable
commandButton	Renders an HTML submit button
commandLink	Renders an HTML link that submits a form
dataTable	Renders an HTML table
form	Renders an HTML form
graphicImage	Renders an HTML image
inputHidden	Renders an HTML hidden form field
inputSecret	Renders an HTML password widget
inputText	Renders an HTML single-line entry widget
inputTextarea	Renders an HTML multi-line entry widget
message	Displays a Faces message
messages	Displays all Faces message
outputFormat	Renders HTML text, substituting parameters
outputLabel	Renders an HTML label
outputLink	Renders an HTML link that does not submit a form
outputText	Renders text

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HTML Tags, cont'd

Tag	Summary
panelGrid	Renders and HTML table
panelGroup	Groups elements within a panel
selectBooleanCheckbox	Renders an HTML checkbox
selectManyCheckbox	Renders a group of HTML checkboxes
selectManyListbox	Renders an HTML listbox
selectManyMenu	Renders an HTML drop-down list
selectOneListbox	Renders an HTML listbox
selectOneMenu	Renders an HTML drop-down list
selectOneRadio	Renders an HTML radio-button group

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Common Attributes

- Many of the tags share common attributes

Property	Type	Default	Required?	Description
binding	String	None	No	Expression that associates this component with a property in a "backing bean"
id	String	None	Yes	The component's unique ID
converter	Object	None	No	Assigns a converter
rendered	boolean	true	No	Display component?
styleClass	String	None	No	CSS style class
value	Object	None	Yes	Text (could be expression)

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HTML Pass Through Attributes

- In general, in addition to the attributes defined by the tag, HTML components support basic HTML attributes such as **cols** and **rows** on an **inputTextarea** tag

```
<h:inputTextarea id="mytextarea" cols="30" rows="12"
   value="#{student1.comments}"/>

<h:outputText style="color:red"
   value="#{header['user-agent']}">
```

See any good HTML reference for a description of these passthrough attributes.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

HTML Output Tags

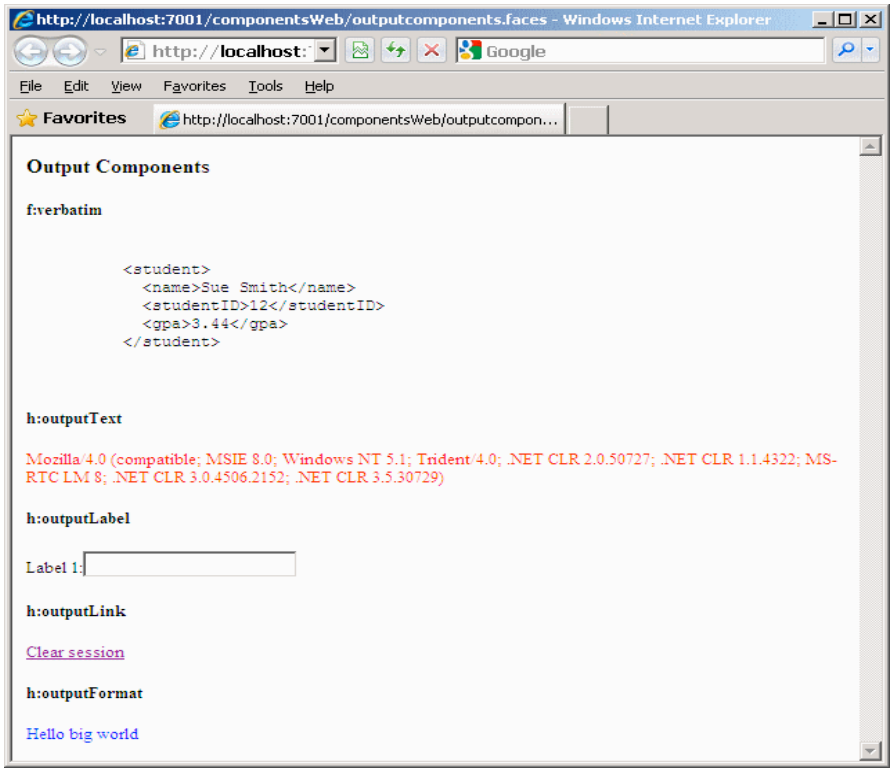
- Output tags create components that generate HTML output of some sort
- Note that **h:outputLink** is intended to hyperlink to a non-JSF resource; use **h:commandLink** to involve JSF

Action	Component	Description
h:outputFormat	HtmlOutputFormat	Renders text with substitution parameters
h:outputLabel	HtmlOutputLabel	Renders an HTML label
h:outputLink	HtmlOutputLink	Renders an HTML hyperlink
h:outputText	HtmlOutputText	Renders HTML text
f:verbatim	UIOutput	Renders verbatim text

The h:outputLink action can contain nested f:param elements that act as URL query parameters. Also note that this action calls encodeResourceURL to format parameters as required by HTTP.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Output Tags Example



outputcomponents.xhtml

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4    <html xmlns="http://www.w3.org/1999/xhtml"
5        xmlns:ui="http://java.sun.com/jsf/facelets"
6        xmlns:h="http://java.sun.com/jsf/html"
7        xmlns:f="http://java.sun.com/jsf/core">
8      <h:body>
9            <h3>Output Components</h3>
10         <h4>f:verbatim</h4>
11         <pre>
12           <f:verbatim escape="true">
13             <student>
14               <name>Sue Smith</name>
15               <studentID>12</studentID>
16               <gpa>3.44</gpa>
17             </student>
18           </f:verbatim>
19         </pre>
20         <h4>h:outputText</h4>
21         <h:outputText style="color:red" value="#{header['user-agent']}">
22         </h:outputText>
23         <h4>h:outputLabel</h4>
24         <h:outputLabel for="field1">Label 1:</h:outputLabel>
25         <h:inputText id="field1"/>
26         <h4>h:outputLink</h4>
27         <h:outputLink value="clearsession.jsp" >
28           Clear session
29           <f:param name="myparam" value="12"/>
30         </h:outputLink>
31         <h4>h:outputFormat</h4>
32         <h:outputFormat style="color:blue" value="Hello {0} world">
33           <f:param value="big" />
34         </h:outputFormat>
35
36     </h:body>
37
38   </html>
```

The h:form Tag

- The **h:form** tag generates an HTML *form* element
- Typically, buttons and other input components should be contained in a form

```
1    <html>
2        <h:body>
3         <h:form>
4
5           <h:commandButton
6             action="#{student1.updateStudent}"
7             value="Submit"/>
8
9         </h:form>
10      </h:body>
11   </html>
```


Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

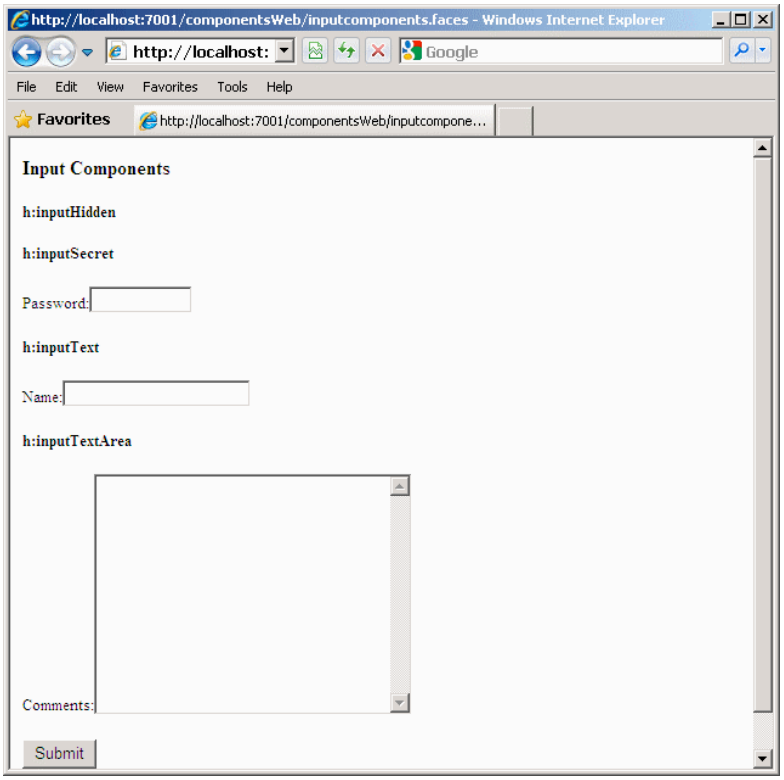
HTML Input Tags

- Most of these tags create components that generate HTML widgets that reside in a form and let the user input text
- The **h:hidden** tag generates an HTML hidden form field

Action	Component	Description
h:inputHidden	HtmlInputHidden	Renders an HTML hidden form field
h:inputSecret	HtmlInputSecret	Renders an HTML password widget
h:inputText	HtmlInputText	Renders an HTML single-line input widget
h:inputTextarea	HtmlInputTextarea	Renders an HTML multi-line input widget

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Input Tags Example



inputcomponents.xhtml

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4    <html xmlns="http://www.w3.org/1999/xhtml"
5        xmlns:ui="http://java.sun.com/jsf/facelets"
6        xmlns:h="http://java.sun.com/jsf/html"
7        xmlns:f="http://java.sun.com/jsf/core">
8      <h:body>
9          <h3>Input Components</h3>
10         <h:form>
11           <h4>h:inputHidden</h4>
12           <h:inputHidden id="myhidden" value="#{student1.gpa}"/>
13
14           <h4>h:inputSecret</h4>
15           <h:outputLabel for="mysecret" value="Password:"/>
16           <h:inputSecret id="mysecret" size="10"
17               value="#{student1.password}"/>
18
19           <h4>h:inputText</h4>
20           <h:outputLabel for="mytext" value="Name:"/>
21           <h:inputText id="mytext" value="#{student1.name}"
22             size="20"/>
23
24           <h4>h:inputTextArea</h4>
25           <h:outputLabel for="mytextarea" value="Comments:"/>
26           <h:inputTextarea id="mytextarea" cols="30" rows="12"
27             value="#{student1.comments}"/>
28
29           <p>
30             <h:commandButton action="#{student1.updateStudent}"
31               value="Submit"/>
32           </p>
33         </h:form>
34
35     </h:body>
36
37   </html>
```

HTML Command Tags

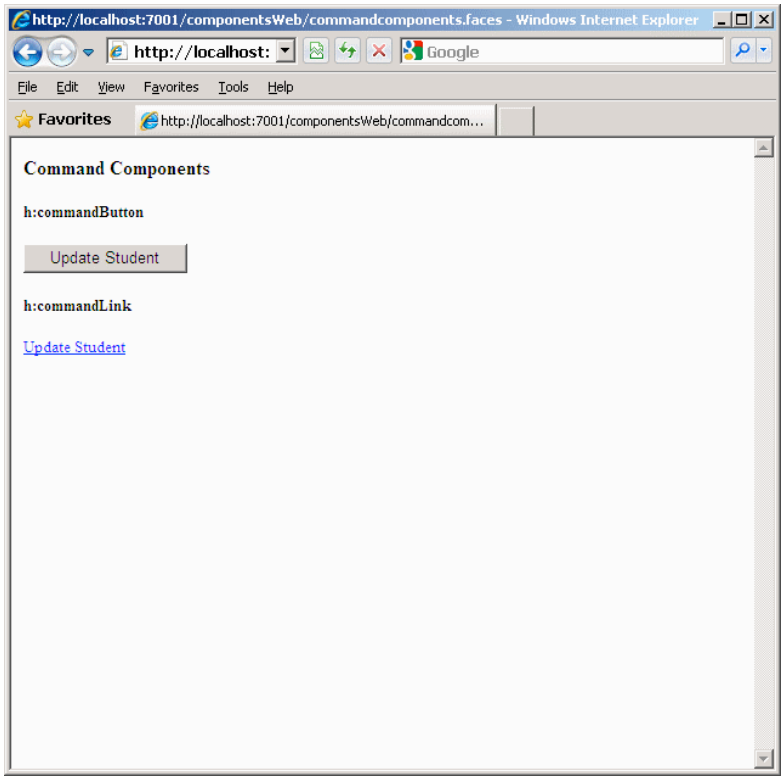
- Command tags create components that generate HTML widgets to let the user submit a form or hyperlink to another page
- The **h:commandLink** action is different from **h:outputLink** since it submits a form

Action	Component	Description
h:commandButton	HtmlCommandButton	Renders an HTML submit button
h:commandLink	HtmlCommandLink	Renders an HTML hyperlink that submits a form

The h:commandButton component has a "type" attribute that defaults to "submit". You can change it to "button" if you want a non-submit button.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Command Tags Example



commandcomponents.xhtml

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4    <html xmlns="http://www.w3.org/1999/xhtml"
5        xmlns:ui="http://java.sun.com/jsf/facelets"
6        xmlns:h="http://java.sun.com/jsf/html"
7        xmlns:f="http://java.sun.com/jsf/core">
8
9      <h:body>
10
11        <h3>Command Components</h3>
12         <h:form>
13           <h4>h:commandButton</h4>
14           <h:commandButton action="#{student1.updateStudent}"
15               value="Update Student"/>
16
17           <h4>h:commandLink</h4>
18           <h:commandLink action="#{student1.updateStudent}"
19             value="Update Student"/>
20         </h:form>
21
22     </h:body>
23
24   </html>
```

Single Selection Components

- These tags create components that generate HTML widgets that let the user choose a single choice from a list

Action	Component	Description
h:selectOneRadio	HtmlSelectOneRadio	Renders an HMTL radio button group
h:selectOneListbox	HtmlSelectOneListbox	Renders an HTML listbox
h:selectOneMenu	HtmlSelectOneMenu	Renders an HTML drop-down list

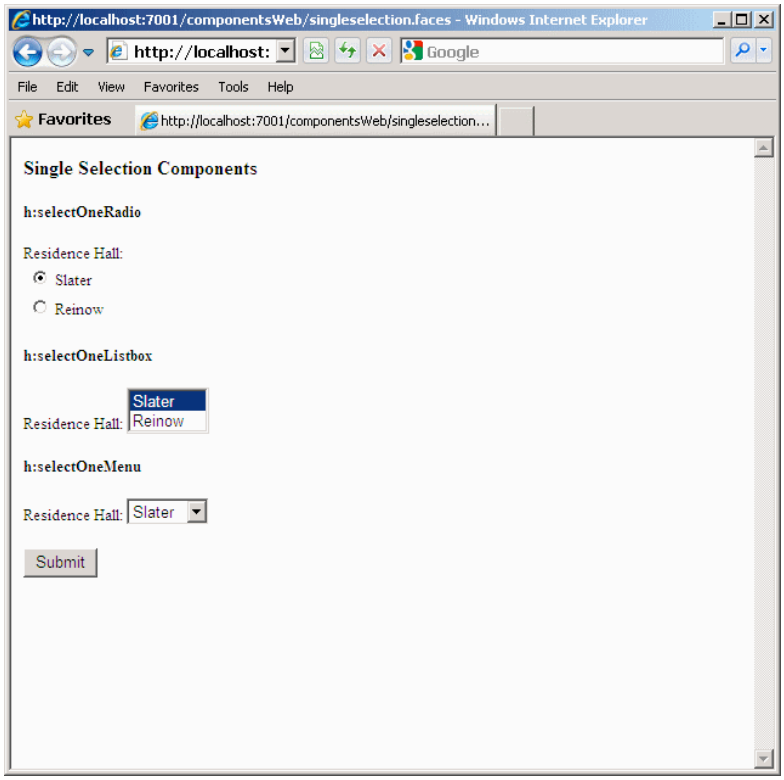
The h:selectOneRadio action accepts a layout attribute with possible values of lineDirection (default, horizontal) or pageDirection (vertical).

The main difference between h:selectOneListbox and h:selectOneMenu is that listboxes let you set a size attribute that controls how many items are visible. In a h:selectOneMenu, the size is fixed at one and thus displays a drop-down list.

Note that the initial selection of these components is determined by the initial value of the associated managed bean property.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Single Selection Example



singleselection.xhtml

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4    <html xmlns="http://www.w3.org/1999/xhtml"
5        xmlns:ui="http://java.sun.com/jsf/facelets"
6        xmlns:h="http://java.sun.com/jsf/html"
7        xmlns:f="http://java.sun.com/jsf/core">
8
9       <h:body>
10
11         <h3>Single Selection Components</h3>
12         <h:form>
13           <h4>h:selectOneRadio</h4>
14           <h:outputLabel for="myselectoneradio"
15              value="Residence Hall: "/>
16           <h:selectOneRadio id="myselectoneradio" layout="pageDirection"
17             value="#{student1.residenceHall}">
18             <f:selectItem itemLabel="Slater" itemValue="Slater"/>
19             <f:selectItem itemLabel="Reinow" itemValue="Reinow"/>
20           </h:selectOneRadio>
21
22           <h4>h:selectOneListbox</h4>
23           <h:outputLabel for="myselectonelistbox"
24              value="Residence Hall: "/>
25           <h:selectOneListbox id="myselectonelistbox" size="2"
26             value="#{student1.residenceHall}">
27             <f:selectItem itemLabel="Slater" itemValue="Slater"/>
28             <f:selectItem itemLabel="Reinow" itemValue="Reinow"/>
29           </h:selectOneListbox>
30
31           <h4>h:selectOneMenu</h4>
32           <h:outputLabel for="myselectonemenu"
33              value="Residence Hall: "/>
34           <h:selectOneMenu id="myselectonemenu"
35             value="#{student1.residenceHall}">
36             <f:selectItem itemLabel="Slater" itemValue="Slater"/>
37             <f:selectItem itemLabel="Reinow" itemValue="Reinow"/>
38           </h:selectOneMenu>
39
40           <p>
41             <h:commandButton action="#{student1.updateStudent}"
42               value="Submit"/>
43           </p>
44         </h:form>
45
```

```
46      </h:body>
47
48
49  </html>
```

Multiple Selection Tags

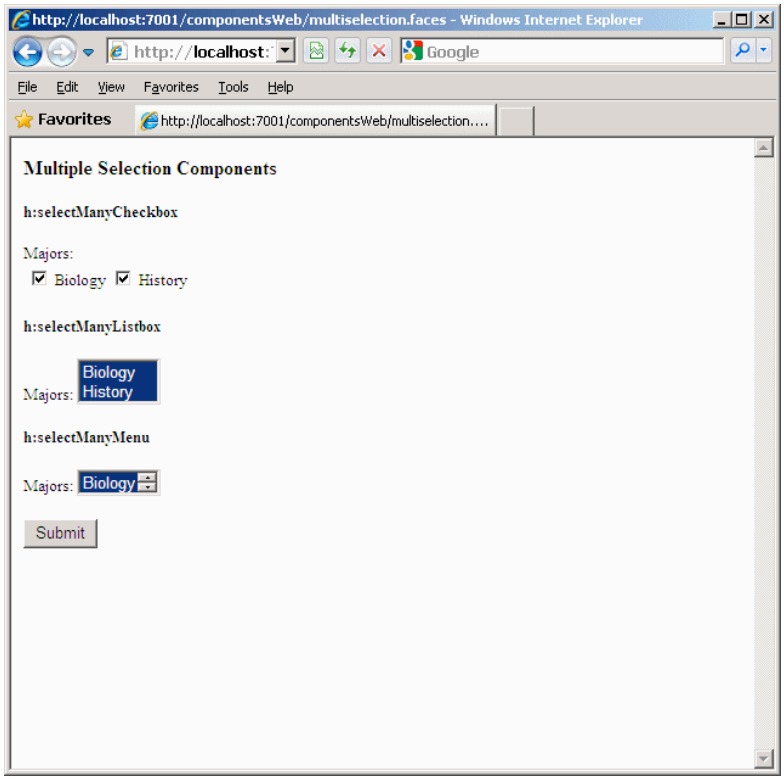
- These tags create components that generate HTML widgets that let the user choose multiple choices from a list
- These components read and write their data to a **List** property in a managed bean

Action	Component	Description
h:selectManyCheckbox	HtmlSelectManyCheckbox	Renders a set of HTML checkboxes
h:selectManyListbox	HtmlSelectManyListbox	Renders an HTML multi-selection list
h:selectManyMenu	HtmlSelectManyMenu	Renders an HTML multi-selection drop-down list

The difference between listboxes and menus is the same as for single-selection components.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Multiple Selection Example



multiselection.xhtml

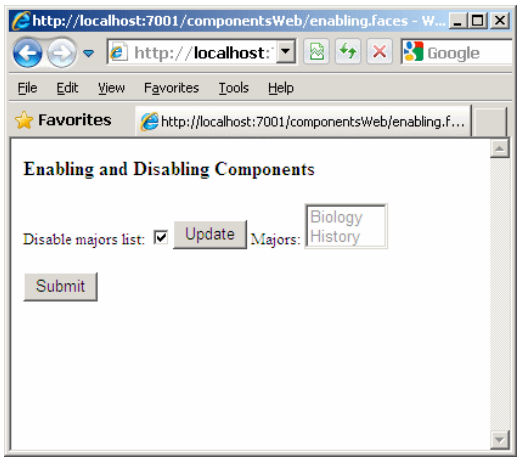
Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4    <html xmlns="http://www.w3.org/1999/xhtml"
5        xmlns:ui="http://java.sun.com/jsf/facelets"
6        xmlns:h="http://java.sun.com/jsf/html"
7        xmlns:f="http://java.sun.com/jsf/core">
8
9      <h:body>
10
11         <h3>Multiple Selection Components</h3>
12         <h:form>
13           <h4>h:selectManyCheckbox</h4>
14           <h:outputLabel for="myselectmanycheckbox"
15              value="Majors: "/>
16           <h:selectManyCheckbox id="myselectmanycheckbox" layout="lineDirection"
17             value="#{student1.majors}">
18             <f:selectItem itemLabel="Biology" itemValue="Biology"/>
19             <f:selectItem itemLabel="History" itemValue="History"/>
20           </h:selectManyCheckbox>
21
22
23           <h4>h:selectManyListbox</h4>
24           <h:outputLabel for="myselectmanylistbox"
25              value="Majors: "/>
26           <h:selectManyListbox id="myselectmanylistbox" size="2"
27             value="#{student1.majors}">
28             <f:selectItem itemLabel="Biology" itemValue="Biology"/>
29             <f:selectItem itemLabel="History" itemValue="History"/>
30           </h:selectManyListbox>
31
32           <h4>h:selectManyMenu</h4>
33           <h:outputLabel for="myselectmanymenu"
34              value="Majors: "/>
35           <h:selectManyMenu id="myselectmanymenu"
36             value="#{student1.majors}">
37             <f:selectItem itemLabel="Biology" itemValue="Biology"/>
38             <f:selectItem itemLabel="History" itemValue="History"/>
39           </h:selectManyMenu>
40
41
42           <p>
43             <h:commandButton action="#{student1.updateStudent}"
44               value="Submit"/>
45           </p>
```

```
46          </h:form>
47
48
49      </h:body>
50
51  </html>
```

Conditionally Enabling Components

- In some cases, you may want to enable and disable components based on application state or user interaction



enabling.xhtml

In this example, we have included a command button to apply the "disable majors list" setting since clicking a checkbox does NOT cause a page submission by default.

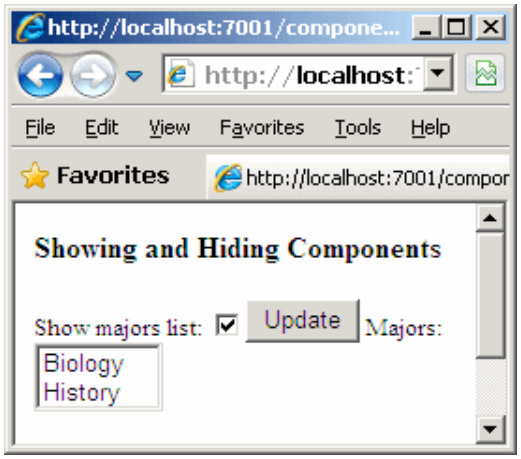
Note that the application could be more responsive if we used client-side scripting to submit the setting whenever the user changes the checkbox.

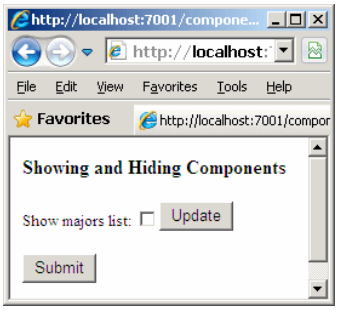
Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4    <html xmlns="http://www.w3.org/1999/xhtml"
5        xmlns:ui="http://java.sun.com/jsf/facelets"
6        xmlns:h="http://java.sun.com/jsf/html"
7        xmlns:f="http://java.sun.com/jsf/core">
8
9      <h:body>
10
11         <h3>Enabling and Disabling Components</h3>
12         <h:form>
13           <h:outputLabel for="mycheckbox"
14              value="Disable majors list: "/>
15           <h:selectBooleanCheckbox id="mycheckbox"
16             value="#{studentBackingBean.showMajors}"/>
17           <h:commandButton action="#{studentBackingBean.update}"
18             value="Update"/>
19
20           <h:outputLabel for="myselectmanylistbox"
21              value="Majors: "/>
22           <h:selectManyListbox id="myselectmanylistbox" size="2"
23             value="#{student1.majors}" disabled="#{studentBackingBean.showMajors}">
24             <f:selectItem itemLabel="Biology" itemValue="Biology"/>
25             <f:selectItem itemLabel="History" itemValue="History"/>
26           </h:selectManyListbox>
27
28           <p>
29             <h:commandButton action="#{student1.updateStudent}"
30               value="Submit"/>
31           </p>
32         </h:form>
33
34     </h:body>
35
36   </html>
```

Conditionally Displaying Components

- In some cases, you may wish to show and hide components based on application state or user interaction





hiding.jsp

In this example, we have included a command button to apply the "show majors list" setting since clicking a checkbox does NOT cause a page submission by default.

Note that the application could be more responsive if we used client-side scripting to submit the setting whenever the user changes the checkbox.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
2    <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
3    <html>
4      <f:view>
5        <body>
6          <h3>Showing and Hiding Components</h3>
7          <h:form>
8            <h:outputLabel for="mycheckbox"
9               value="Show majors list: "/>
10           <h:selectBooleanCheckbox id="mycheckbox"
11             value="#{studentBackingBean.showMajors}"/>
12           <h:commandButton action="#{studentBackingBean.update}"
13             value="Update"/>
14
15           <h:outputLabel for="myselectmanylistbox"
16              value="Majors: " rendered="#{studentBackingBean.showMajors}"/>
17           <h:selectManyListbox id="myselectmanylistbox" size="2"
18             value="#{student1.majors}" rendered="#{studentBackingBean.showMajors}">
19             <f:selectItem itemLabel="Biology" itemValue="Biology"/>
20             <f:selectItem itemLabel="History" itemValue="History"/>
21           </h:selectManyListbox>
22
23           <p>
24             <h:commandButton action="#{student1.updateStudent}"
25               value="Submit"/>
26           </p>
27         </h:form>
28       </body>
29     </f:view>
30   </html>
```

Chapter Summary

In this chapter, you learned:

- How to create components on JSF pages using JSF tags
- About different categories of JSF components

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Conversion and Validation

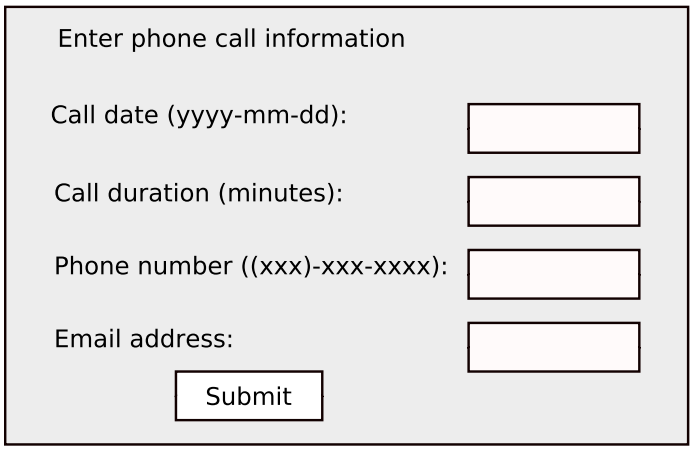
- Converters
- Validators

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Data Conversion

- In a Web-based system, the user enters data as character strings, but applications typically used data types such as **double** or **Date**
- JSF provides facilities to convert types and to define and display conversion error messages

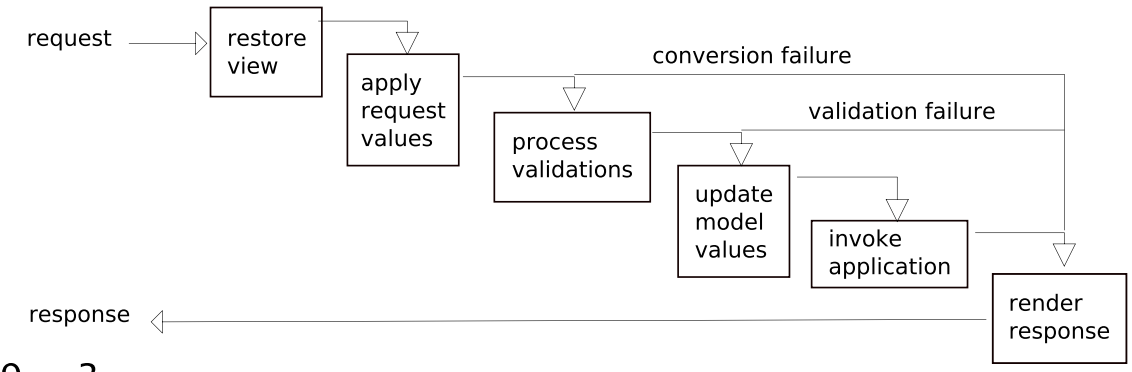


The Web uses HTTP, and HTTP passes all request parameter values as character strings.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Request Lifecycle

- JSF components attempt to convert data during the **Process Validations** phase
- If conversion (or validation) fails, JSF queues one or more error message strings and posts back to the same page
- The page should display the error message strings so that the user can correct their input



JSF provides custom actions (tags) to display the message strings.

Note that if conversion or validation fails, JSF does not invoke the application.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Standard Converters

Converter Name	Converter Tag	Type(s) Converted
javax.faces.DateTime	f:convertDateTime	java.util.Date
javax.faces.Number	f:convertNumber	java.lang.Long java.lang.Double
javax.faces.Boolean	n/a	boolean,java.lang.Boolean
javax.faces.Byte	n/a	byte,java.lang.Byte
javax.faces.Character	n/a	char,java.lang.Character
javax.faces.Double	n/a	double,java.lang.Double
javax.faces.Float	n/a	float,java.lang.Float
javax.faces.Integer	n/a	int,java.lang.Integer
javax.faces.Long	n/a	long,java.lang.Long
javax.faces.Short	n/a	short,java.lang.Short
javax.faces.BigDecimal	n/a	java.math.BigDecimal
javax.faces.BigInteger	n/a	java.math.BigInteger

Each converter in JSF has a symbolic name and works on the specified types. In other words, if a managed bean has a property of type 'short' and a component binds to that property, JSF will automatically select the javax.faces.Short converter.

Dates require more information for conversion, so there's a special tag for them. In addition, numbers sometimes have more particular conversion requirements (e.g minimum number of fractional digits), so JSF provides an action for numbers, too.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Converter Example

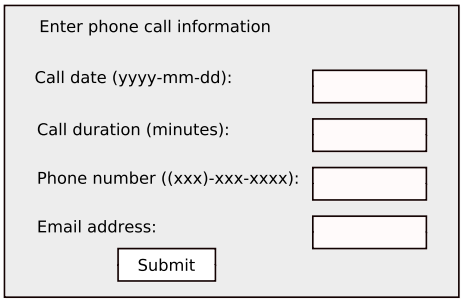
callinfo.jsp

```
<h:inputText id="duration"
   value="#{callinfo1.callLength}"/>
```

JSF chooses the javax.faces.Integer converter

CallInfo.java
managed bean

```
public class CallInfo
{
   . . .
   public int getCallLength() {}
   public void setCallLength(int callLength) {}
}
```



In this example, since the "callLength" property type is "int", JSF can use the standard javax.faces.Integer converter.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Converting Dates

- If the property specified in a value-binding expression is defined with a primitive type, JSF can choose the appropriate converter automatically
- But dates require special treatment since they can be represented in many different forms, so you must specify how dates should be converted

callinfo.jsp

```
<h:inputText id="calldate"
  value="#{callinfo1.callDate}">
  <f:convertDateTime type="date"
    pattern="yyyy-MM-dd"/>
</h:inputText>
```

JSF uses the javax.faces.DateTime converter

CallInfo.java
managed bean

```
import java.util.Date;
public class CallInfo
{
   . . .
   public Date getCallDate() {}
   public void setCallDate(Date d) {}
}
```


It's also important to note that dates have different representations based on human cultures. We will cover more about that topic in the Internationalization chapter.

Also note that the f:convertDateTime tag has several other options shown here. For example, instead of specifying a date pattern, you could specify:

```
<f:convertDateTime dateStyle="short" />
```

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Displaying Conversion Error Messages

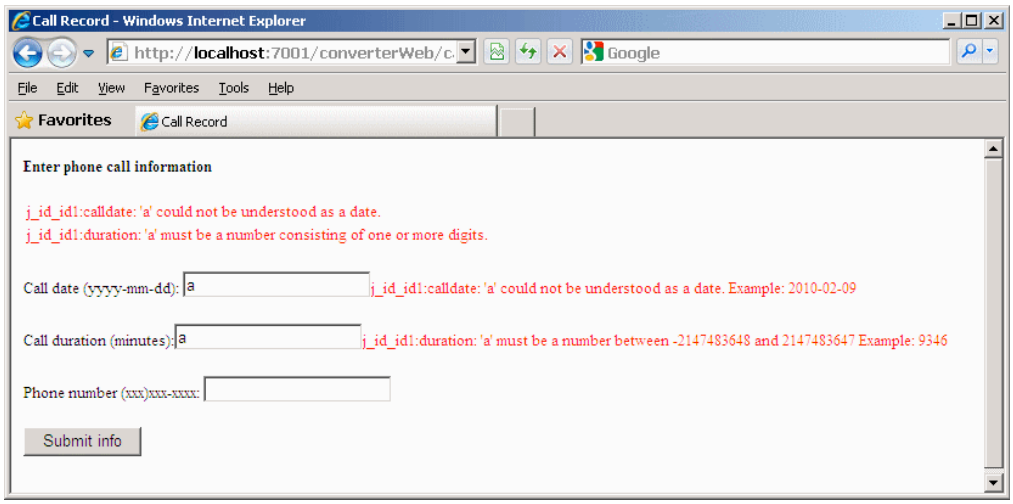
- JSF provides the **h:messages** and **h:message** actions that let a JSF display error messages

```
1    . . .
2    <h:form>
3      <h:messages layout="table" style="color:red"/>
4      <p>
5        <h:outputLabel for="calldate"
6          value="Call date (yyyy-mm-dd): "/>
7        <h:inputText id="calldate"
8          value="#{callinfo1.callDate}">
9           <f:convertDateTime type="date"
10             pattern="yyyy-MM-dd"/>
11       </h:inputText>
12       <h:message for="calldate" style="color:red"/>
13     </p>
14   . . .
```


Most JSF applications use either h:messages to display a message list or multiple h:message actions, one for each input component.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Conversion Error Messages, cont'd



Here we show the converters in action. Note that the standard error messages are less than pleasing.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Custom Converter Error Messages

- JSF converters define standard messages, but you can override those by defining message strings in a **resource file**
- The standard converters support both **summary** and **detail** messages
- You must configure the resource file in faces-config.xml

messages.properties

```
javax.faces.converter.IntegerConverter.INTEGER_detail=Please enter a number
javax.faces.converter.IntegerConverter.INTEGER=The value ''{0}'' \
 is incorrect. It should be a number
```

faces-config.xml

```
<faces-config . . .>
 <application>
  <message-bundle>messages</message-bundle>
 </application>
  . . .
</faces-config>
```

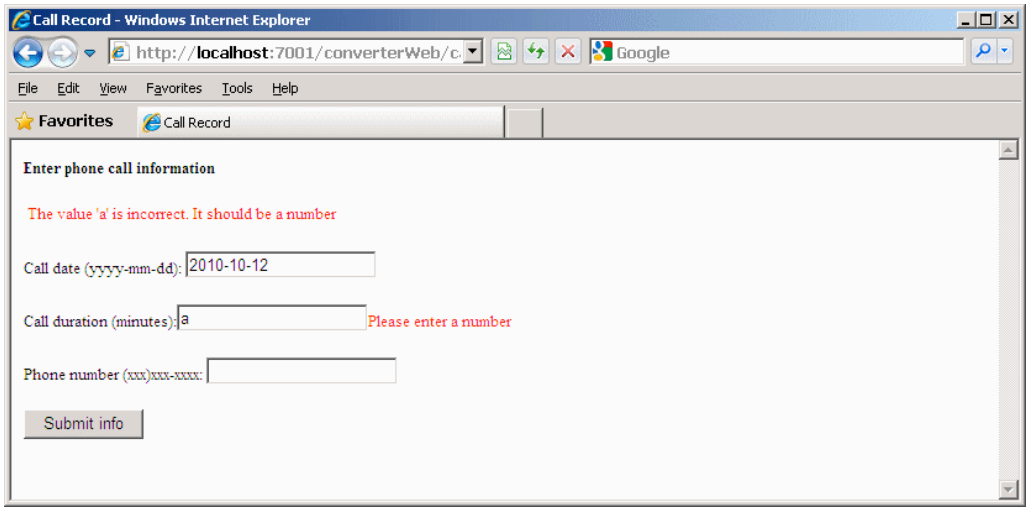

Note that each converter has two "keys" in the resource file. By default, the h:messages action displays the non-detail string, while h:messages displays the detail string. You can override that behavior in either case by writing the appropriate attributes, for example:

```
<h:message for="duration" showSummary="true"
    showDetail="false"
    style="color:red"/>
```

NOTE: In the non-detail message shown here, it looks like there are double-quotes around the {0}, but they are actually two single quotes on each side. Properties files have an odd syntax for strings that include placeholders and use single quotes -- you must repeat the single quotes.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Custom Converter Error Messages, cont'd



Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Custom Converter Error Messages, cont'd

- Starting in JSF 1.2, there's a simple way to display custom converter messages without creating a resource file

```
1    <h:inputText id="duration"
2       value="#{callinfo1.callLength}"
3       converterMessage="Duration messed up"/>
4    <h:message for="duration" style="color:red"/>
```


Though this is simpler, there's one possible downside: you can't use substitution "{x}" placeholders with this approach.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Standard Converter and Validation Messages

standard-conversion-strings.txt

This info is taken from the JSF specification.

Note that several of the strings have {x} placeholders into which the converter can insert text.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
javax.faces.component.UIInput.CONVERSION -- {0}: Conversion error occurred
javax.faces.component.UIInput.REQUIRED -- {0}: Validation Error: Value is
  required
javax.faces.component.UIInput.UPDATE -- {0}: An error occurred when
  processing your submitted information
javax.faces.component.UISelectOne.INVALID -- {0}: Validation Error: Value is
  not valid
javax.faces.component.UISelectMany.INVALID -- {0}: Validation Error: Value is
  not valid
javax.faces.converter.BigDecimalConverter.DECIMAL={2}: ''{0}'' must be a
  signed decimal number.
javax.faces.converter.BigDecimalConverter.DECIMAL_detail={2}: ''{0}'' must be
  a signed decimal number consisting of zero or more digits, that may be
  followed by a decimal point and fraction. Example: {1}
javax.faces.converter.BigIntegerConverter.BIGINTEGER={2}: ''{0}'' must be a
  number consisting of one or more digits.
javax.faces.converter.BigIntegerConverter.BIGINTEGER_detail={2}: ''{0}''
  must be a number consisting of one or more digits. Example: {1}
javax.faces.converter.BooleanConverter.BOOLEAN={1}: ''{0}'' must be 'true'
  or 'false'.
javax.faces.converter.BooleanConverter.BOOLEAN_detail={1}: ''{0}'' must be
  'true' or 'false'. Any value other than 'true' will evaluate to 'false'.
javax.faces.converter.ByteConverter.BYTE={2}: ''{0}'' must be a number between
  0 and 255.
javax.faces.converter.ByteConverter.BYTE_detail={2}: ''{0}'' must be a number
  between 0 and 255. Example: {1}
javax.faces.converter.CharacterConverter.CHARACTER={1}: ''{0}'' must be a valid
  character.
javax.faces.converter.CharacterConverter.CHARACTER_detail={1}: ''{0}'' must be
  a valid ASCII character.
javax.faces.converter.DateTimeConverter.DATE={2}: ''{0}'' could not be
  understood as a date.
javax.faces.converter.DateTimeConverter.DATE_detail={2}: ''{0}'' could
  not be understood as a date. Example: {1}
javax.faces.converter.DateTimeConverter.TIME={2}: ''{0}'' could not be
  understood as a time.
javax.faces.converter.DateTimeConverter.TIME_detail={2}: ''{0}'' could not
  be understood as a time. Example: {1}
javax.faces.converter.DateTimeConverter.DATETIME={2}: ''{0}'' could not be
  understood as a date and time.
javax.faces.converter.DateTimeConverter.DATETIME_detail={2}: ''{0}'' could
  not be understood as a date and time. Example: {1}
javax.faces.converter.DateTimeConverter.PATTERN_TYPE={1}: A 'pattern' or
  'type' attribute must be specified to convert the value ''{0}''.
javax.faces.converter.DoubleConverter.DOUBLE={2}: ''{0}'' must be a number
```

```
  consisting of one or more digits.
javax.faces.converter.DoubleConverter.DOUBLE_detail={2}: ''{0}'' must be
  a number between 4.9E-324 and 1.7976931348623157E308 Example: {1}
javax.faces.converter.EnumConverter.ENUM={2}: ''{0}'' must be convertible
  to an enum.
javax.faces.converter.EnumConverter.ENUM_detail={2}: ''{0}'' must be
  convertible to an enum from the enum that contains the constant ''{1}''.
javax.faces.converter.EnumConverter.ENUM_NO_CLASS={1}: ''{0}'' must be
  convertible to an enum from the enum, but no enum class provided.
javax.faces.converter.EnumConverter.ENUM_NO_CLASS_detail={1}: ''{0}''
  must be convertible to an enum from the enum, but no enum class provided.
javax.faces.converter.FloatConverter.FLOAT={2}: ''{0}'' must be a number
  consisting of one or more digits.
javax.faces.converter.FloatConverter.FLOAT_detail={2}: ''{0}'' must be a
  number between 1.4E-45 and 3.4028235E38 Example: {1}
javax.faces.converter.IntegerConverter.INTEGER={2}: ''{0}'' must be a
  number consisting of one or more digits.
javax.faces.converter.IntegerConverter.INTEGER_detail={2}: ''{0}'' must
  be a number between -2147483648 and 2147483647 Example: {1}
javax.faces.converter.LongConverter.LONG={2}: ''{0}'' must be a number
  consisting of one or more digits.
javax.faces.converter.LongConverter.LONG_detail={2}: ''{0}'' must be a
  number between -9223372036854775808 to 9223372036854775807 Example: {1}
javax.faces.converter.NumberConverter.CURRENCY={2}: ''{0}'' could not
  be understood as a currency value.
javax.faces.converter.NumberConverter.CURRENCY_detail={2}: ''{0}'' could
  not be understood as a currency value. Example: {1}
javax.faces.converter.NumberConverter.PERCENT={2}: ''{0}'' could not be
  understood as a percentage.
javax.faces.converter.NumberConverter.PERCENT_detail={2}: ''{0}'' could
  not be understood as a percentage. Example: {1}
javax.faces.converter.NumberConverter.NUMBER={2}: ''{0}'' is not a number.
javax.faces.converter.NumberConverter.NUMBER_detail={2}: ''{0}'' is not
  a number. Example: {1}
javax.faces.converter.NumberConverter.PATTERN={2}: ''{0}'' is not a
  number pattern.
javax.faces.converter.NumberConverter.PATTERN_detail={2}: ''{0}'' is not
  a number pattern. Example: {1}
javax.faces.converter.ShortConverter.SHORT={2}: ''{0}'' must be a number
  consisting of one or more digits.
javax.faces.converter.ShortConverter.SHORT_detail={2}: ''{0}'' must be a
  number between -32768 and 32767 Example: {1}
javax.faces.converter.STRING={1}: Could not convert ''{0}'' to a string.
javax.faces.validator.DoubleRangeValidator.MAXIMUM -- {1}: Validation
  Error: Value is greater than allowable maximum of ''{0}''
```

```
javax.faces.validator.DoubleRangeValidator.MINIMUM -- {1}: Validation
  Error: Value is less than allowable minimum of ''{0}''
javax.faces.validator.DoubleRangeValidator.NOT_IN_RANGE -- {2}:
  Validation Error: Specified attribute is not between the expected
  values of {0} and {1}.
javax.faces.validator.DoubleRangeValidator.TYPE -- {0}: Validation
  Error: Value is not of the correct type
javax.faces.validator.LengthValidator.MAXIMUM -- {1}: Validation Error:
  Value is greater than allowable maximum of ''{0}''
javax.faces.validator.LengthValidator.MINIMUM -- {1}: Validation Error:
  Value is less than allowable minimum of ''{0}''
javax.faces.validator.LongRangeValidator.MAXIMUM -- {1}: Validation
  Error: Value is greater than allowable maximum of ''{0}''
javax.faces.validator.LongRangeValidator.MINIMUM -- {1}: Validation Error
  Value is less than allowable minimum of ''{0}''
javax.faces.validator.LongRangeValidator.NOT_IN_RANGE={2}: Validation
  Error: Specified attribute is not between the expected values of
  {0} and {1}.
javax.faces.validator.LongRangeValidator.TYPE -- {0}: Validation Error:
  Value is not of the correct type
```

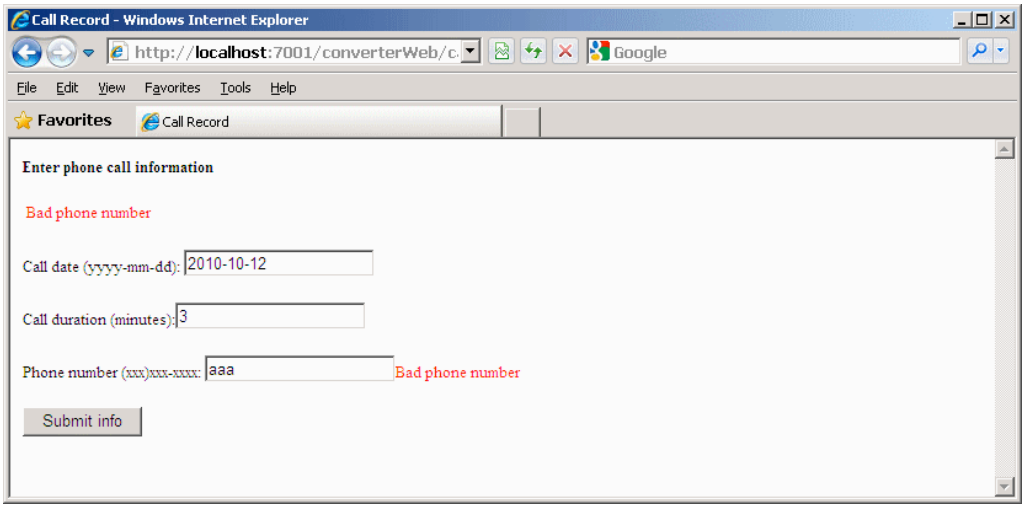
Custom Converters

- To write a converter for data types not covered by one of the standard converters:
    - Write a class that implements the **Converter** interface
    - Register the converter in faces-config.xml
    - Configure components to use the custom converter (optional)

When you register the converter, you can optionally specify the class that the converter works on. If you do that, then you don't need to configure components to use the custom converter, since JSF will find the converter via the property type. If you don't specify a class-to-convert, then you should configure the converter with an ID that you can specify when you configure a component to use the converter.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Custom Converter Example



PhoneNumberConverter.java

PhoneNumber.java

callrecord.jsp

faces-config.xml

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package converters;
2
3    import javax.faces.application.FacesMessage;
4    import javax.faces.component.UIComponent;
5    import javax.faces.context.FacesContext;
6    import javax.faces.convert.Converter;
7    import javax.faces.convert.ConverterException;
8    import callcenter.PhoneNumber;
9
10   public class PhoneNumberConverter implements Converter
11   {
12       @Override
13       public Object getAsObject(FacesContext ctx,
14               UIComponent component, String val)
15       {
16           return convert(val);
17       }
18
19       private PhoneNumber convert(String s)
20       {
21           if (s == null || s.length() == 0)
22               return null;
23
24           String phoneNumberPattern =
25               "\\(\\d{3}\\)\\d{3}-\\d{4}";
26           boolean ok = s.matches(phoneNumberPattern);
27           if (!ok)
28               throw new ConverterException(new FacesMessage("Bad phone number"));
29
30           int acEnd = s.indexOf(')');
31           String number = s.substring(acEnd+1);
32           String areaCode = s.substring(1, acEnd);
33
34           return new PhoneNumber(areaCode, number);
35       }
36
37       @Override
38       public String getAsString(FacesContext ctx,
39               UIComponent component, Object o)
40       {
41           return o.toString();
42       }
43   }
```

```
1    <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
2    <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
3    <html>
4      <head>
5        <title>Call Record</title>
6      </head>
7      <f:view>
8        <body>
9          <h4>Enter phone call information</h4>
10         <h:form>
11           <h:messages layout="table" style="color:red"/>
12           <p>
13             <h:outputLabel for="calldate" value="Call date (yyyy-mm-dd): "/>
14             <h:inputText id="calldate"
15               value="#{callinfo1.callDate}">
16               <f:convertDateTime type="date"
17                 pattern="yyyy-MM-dd"/>
18             </h:inputText>
19             <h:message for="calldate"
20                style="color:red"/>
21           </p>
22           <p>
23             <h:outputLabel for="duration" value="Call duration (minutes):"/>
24             <h:inputText id="duration"
25                value="#{callinfo1.callLength}"/>
26             <h:message for="duration" style="color:red"/>
27           </p>
28           <p>
29             <h:outputLabel for="phonenumber" value="Phone number (xxx)xxx-xxxx: "/>
30             <h:inputText id="phonenumber" value="#{callinfo1.number}">
31               <f:converter converterId="converters.PhoneNumberConverter"/>
32             </h:inputText>
33             <h:message for="phonenumber" style="color:red"/>
34           </p>
35           <p>
36             <h:commandButton value="Submit info"
37               action="#{callinfo1.updateCall}"/>
38           </p>
39         </h:form>
40       </body>
41     </f:view>
42   </html>
```

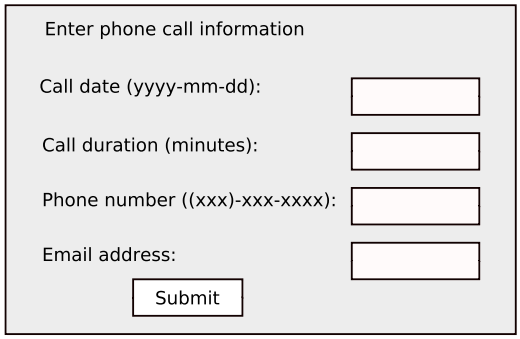
```
1    <?xml version="1.0" encoding="UTF-8"?>
2
3    <faces-config
4        xmlns="http://java.sun.com/xml/ns/javaee"
5        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/
7        version="1.2">
8        <application>
9            <message-bundle>messages</message-bundle>
10       </application>
11       <converter>
12           <display-name>PhoneNumberConverter</display-name>
13           <converter-id>converters.PhoneNumberConverter</converter-id>
14           <converter-class>converters.PhoneNumberConverter</converter-class>
15       </converter>
16       <managed-bean>
17           <managed-bean-name>callinfo1</managed-bean-name>
18           <managed-bean-class>callcenter.CallInfo</managed-bean-class>
19           <managed-bean-scope>session</managed-bean-scope>
20       </managed-bean>
21       <lifecycle>
22           <phase-listener>listeners.MyPhaseListener</phase-listener>
23       </lifecycle>
24
25   </faces-config>
```

```
1    package callcenter;
2
3    public class PhoneNumber
4    {
5        private String areaCode;
6        private String number;
7
8        public PhoneNumber(String number)
9        {
10           this.number = number;
11       }
12
13       public PhoneNumber(String areaCode,
14                String number)
15       {
16           this.areaCode = areaCode;
17           this.number = number;
18       }
19
20       public String getAreaCode()
21       {
22           return areaCode;
23       }
24
25       public void setAreaCode(String areaCode)
26       {
27           this.areaCode = areaCode;
28       }
29
30       public String getNumber()
31       {
32           return number;
33       }
34
35       public void setNumber(String number)
36       {
37           this.number = number;
38       }
39
40       @Override
41       public String toString()
42       {
43           StringBuilder sb = new StringBuilder();
44
45           if (areaCode != null)
```

```
46          {
47              sb.append("(");
48              sb.append(areaCode);
49              sb.append(")");
50          }
51          sb.append(number);
52          return sb.toString();
53      }
54  }
```

What is Validation?

- Validation lets an application verify that users enter proper data, for example that a serial number conforms to a particular pattern or that a quantity is positive number
- JSF applications use **converters** and **validators** to perform this checking



All programs, even standalone programs, need to check the input from users, but validation is especially critical for Web applications. That's because Web apps involve network traffic and are susceptible to attacks by "script kiddies" and others interested in cracking or breaking the system.

So all well designed Web applications should check user input, but note that you really only need to check user input widgets where the user enters data. For example, if a Web page contains a read-only drop-down list, there's no need to validate that entry.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Validation Scenarios

Validation	Description
Required Field	Ensure that user entered data in the field
Data Type	Ensure that user enter correct kind of data (e.g. an integer)
String Length	Ensure that the user entered at least n characters but less than m
Value Range	Ensure that numeric values fall within a range
Pattern Match	Ensure that entered text matches a regular expression

This page lists some of the generic kinds of validation performed by Web applications.

Data type validation is actually done by converters in JSF.

Pattern matching validation with regular expressions is not a standard part of JSF 1.x, but is included in JSF 2. Note that you also write custom validators that use patterns (regular expression) to validate.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Converters vs Validators

- There is some conceptual overlap between converters and validators, since they both examine user input and decide if it's correct
- Converters provide basic checking (for example, data type), while validators provide more extensive checking (e.g. ranges of numeric values)
- Converters also run during the **Render Response** phase, where they convert Java types into strings

In either case, it's possible to write your own custom converter or validator for special situations.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Required Fields

- If you specify the **required=true** attribute on a component, if the user leaves that field empty, JSF will queue an error message and post-back to the original page
- For consistency, JSF2 adds the **f:validateRequired** tag

```
1    <h:inputText id="email" required="true"
2        value="#{callinfo1.email}"/>
```

The message key string for this is:

```
javax.faces.component.UIInput.REQUIRED
```

With a value of:

```
{0}: Validation Error: Value is required
```

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Standard Validators

- JSF provides standard validators that cover many common validation scenarios

Tag	Attributes
f:validateDoubleRange	minimum,maximum
f:validateLength	minimum,maximum
f:valdateLongRange	minimum,maximum
f:validateRequired (JSF2)	
f:validateRegex (JSF2)	pattern

```
1    <h:inputText id="duration"
2       value="#{callinfo1.callLength}"
3       required="true">
4         <f:validateLongRange
5           minimum="1"
6           maximum="120"/>
7    </h:inputText>
```


Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Displaying Validation Error Messages

- As with conversion errors, you can display validation error messages either with **h:message** or **h:messages**
- See an earlier page in this chapter for a complete listing of standard validation and conversion messages

```
1    <h:inputText id="duration"
2       value="#{callinfo1.callLength}"
3       required="true">
4         <f:validateLongRange
5           minimum="1"
6           maximum="120"/>
7    </h:inputText>
8
9    <h:message for="duration" style="color:red"/>
```


Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Custom Validation Error Messages

- JSF validators define standard messages, but you can override those by defining message strings in a **resource file**
- You must configure the resource file in faces-config.xml

messages.properties

```
javax.faces.validator.LongRangeValidator.MINIMUM=Please enter\
 a value greater than {0}
javax.faces.validator.LongRangeValidator.MAXIMUM=Please enter\
 a value less than {0}
javax.faces.validator.LongRangeValidator.NOT_IN_RANGE= Please enter \
 a value between {0} and {1}
javax.faces.validator.LengthValidator.MINIMUM=Please enter  \
 at least {0} characters
javax.faces.validator.LongRangeValidator.TYPE=Please enter \
 a number
javax.faces.component.UIInput.REQUIRED=Entry is required
```

faces-config.xml

```
<faces-config . . .>
 <application>
  <message-bundle>messages</message-bundle>
 </application>
  . . .
</faces-config>
```


This process is similar as we saw for custom conversion messages.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Bean Validation Framework

- JSF2 and later support the standard JEE **Bean Validation Framework**
- The BVF lets you define validation rules directly on beans and then JSF components can "inherit" those rules
- This technique lets you define validation rules in a single location

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

History of Bean Validation

- The Hibernate OR/M framework has long included Hibernate Validators that let you annotate Hibernate entities with constraints
- The Bean Validation standard builds on Hibernate Validators and was released in October of 2009

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using Bean Validation

- If you are using JSF 2.x or later, Bean Validation is automatically enabled and works seamlessly – JSF components will automatically pick up validations defined in beans
- If you are using Bean Validation in a non-Web application, for example, a Swing GUI, you can invoke the Bean Validation programmatically

In JSF2, Bean validation works without further configuration. You can, however, DISABLE bean validation for a JSF component:

```
<h:inputText value="#{sampleBean.userName}">
  <f:validateBean disabled="true" />
</h:inputText>
```

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

A Simple Example: The Bean

```
1    public class Student
2    {
3       @NotNull
4       private int studentID;
5
6       @DecimalMin(0.0)
7       @DecimalMax(4.0)
8       private double gpa;
9
10      @Size(min=1, max=300)
11      private String name;
12
13      . . .
14   }
```

You can put the annotations either on fields as shown here, or on a "get" method.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Bean Validation Annotations

Annotation	Description	Example
@AssertTrue	The boolean must be true	@AssertTrue private boolean f;
@Future	The date must be in the future	@Future private Date tomorrow;
@Min	The integer's minimum value	@Min(1) private int quantity;
@Pattern	The string must match a regular expression	@Pattern(regexp="4[0-9]{15}") private String credCard;

This table does not show all of the constraint annotations. For a more complete listing, see:

http://download.oracle.com/javaee/6/tutorial/doc/gircz.html

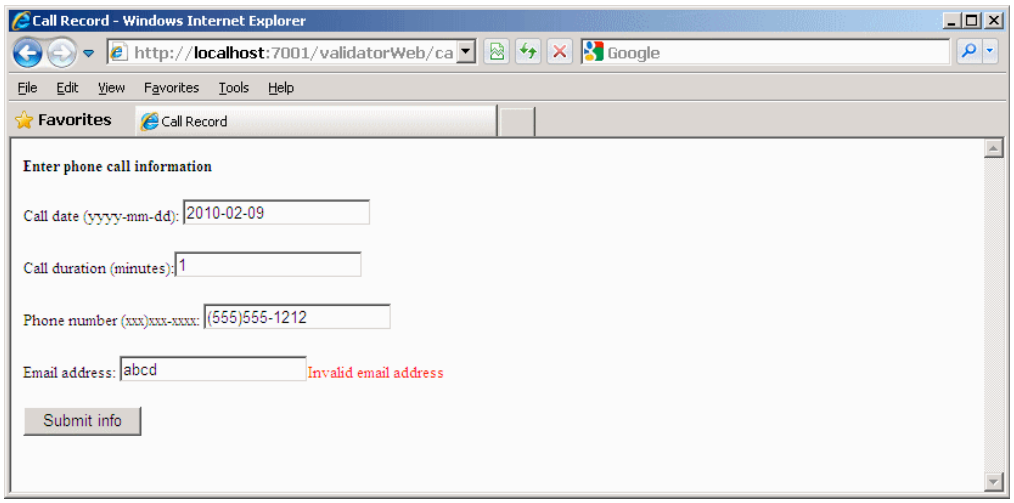
Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining a Custom Validator

- To write a custom validator, follow these steps:
    - Write a class that implements the **Validator** interface
    - Register the validator in faces-config.xml
    - Configure components to use the custom validator

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Custom Validator Example



EmailValidator.java

faces-config-validator.xml

callrecord-validation.jsp

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package validators;
2
3    import javax.faces.application.FacesMessage;
4    import javax.faces.component.UIComponent;
5    import javax.faces.context.FacesContext;
6    import javax.faces.validator.Validator;
7    import javax.faces.validator.ValidatorException;
8
9    public class EmailValidator implements Validator
10   {
11
12       @Override
13       public void validate(FacesContext ctx,
14               UIComponent component, Object value)
15               throws ValidatorException
16       {
17           String emailPattern =
18               "^\\w+@[a-zA-Z_]+?\\.[a-zA-Z]{2,3}$";
19           String email = (String)value;
20
21           boolean ok = email.matches(emailPattern);
22           if (!ok)
23               throw new ValidatorException(
24                 new FacesMessage(
25                   "Invalid email address"));
26       }
27
28   }
```

```
1    <?xml version="1.0" encoding="UTF-8"?>
2
3    <faces-config
4        xmlns="http://java.sun.com/xml/ns/javaee"
5        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/
7        version="1.2">
8        <application>
9            <message-bundle>messages</message-bundle>
10       </application>
11       <converter>
12           <display-name>PhoneNumberConverter</display-name>
13           <converter-id>converters.PhoneNumberConverter</converter-id>
14           <converter-class>converters.PhoneNumberConverter</converter-class>
15       </converter>
16       <managed-bean>
17           <managed-bean-name>callinfo1</managed-bean-name>
18           <managed-bean-class>callcenter.CallInfo</managed-bean-class>
19           <managed-bean-scope>session</managed-bean-scope>
20           <managed-property>
21               <property-name>number</property-name>
22               <property-class>java.lang.String</property-class>
23               <value>(555)555-1212</value>
24           </managed-property>
25           <managed-property>
26               <property-name>callLength</property-name>
27               <property-class>int</property-class>
28               <value>1</value>
29           </managed-property>
30       </managed-bean>
31       <lifecycle>
32           <phase-listener>listeners.MyPhaseListener</phase-listener>
33       </lifecycle>
34       <validator>
35           <display-name>EmailValidator</display-name>
36           <validator-id>validators.EmailValidator</validator-id>
37           <validator-class>validators.EmailValidator</validator-class>
38       </validator>
39
40   </faces-config>
```

```
1    <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
2    <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
3    <html>
4      <head>
5        <title>Call Record</title>
6      </head>
7      <f:view>
8        <body>
9          <h4>Enter phone call information</h4>
10         <h:form>
11           <p>
12             <h:outputLabel for="calldate"
13               value="Call date (yyyy-mm-dd): "/>
14             <h:inputText id="calldate" value="#{callinfo1.callDate}"
15                required="true">
16               <f:convertDateTime type="date" pattern="yyyy-MM-dd"/>
17             </h:inputText>
18             <h:message for="calldate" style="color:red"/>
19           </p>
20           <p>
21             <h:outputLabel for="duration"
22               value="Call duration (minutes):"/>
23             <h:inputText id="duration" value="#{callinfo1.callLength}"
24                 required="true">
25                 <f:validateLongRange minimum="1" maximum="120"/>
26             </h:inputText>
27             <h:message for="duration" style="color:red"/>
28           </p>
29           <p>
30             <h:outputLabel for="phonenumber"
31               value="Phone number (xxx)xxx-xxxx: "/>
32             <h:inputText id="phonenumber" value="#{callinfo1.number}"
33                required="true">
34               <f:converter converterId="converters.PhoneNumberConverter"/>
35             </h:inputText>
36             <h:message for="phonenumber" style="color:red"/>
37           </p>
38           <p>
39             <h:outputLabel for="email"
40               value="Email address: " />
41             <h:inputText id="email"
42                 value="#{callinfo1.email}">
43                 <f:validator validatorId="validators.EmailValidator"/>
44             </h:inputText>
45             <h:message for="email" style="color:red"/>
```

```
46            </p>
47            <p>
48              <h:commandButton value="Submit info"
49                action="#{callinfo1.updateCall}"/>
50            </p>
51          </h:form>
52        </body>
53      </f:view>
54    </html>
```

Bypassing Conversion and Validation

- Sometimes, you need a "submit" button that the user can press without conversion and validation kicking in
- One example is a wizard-type form with a "Cancel" button - the user should not be required to fill in fields if all they want to do is cancel
- Use the **immediate=true** attribute on the command button to skip the **Process Validations** phase

```
<h:commandButton value="Cancel" action="cancel"
  immediate="true"/>
```


Normally, when the user presses a command button, JSF runs the Process Validations phase - if there are conversion or validation errors, JSF redisplays the original page. Using this attribute causes JSF to invoke the application immediately after the Apply Request Values phase, skipping conversion and validation.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- How JSF uses converters to convert user-entered strings into Java data types
- How to use validators to further check user-entered data

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Event Handling

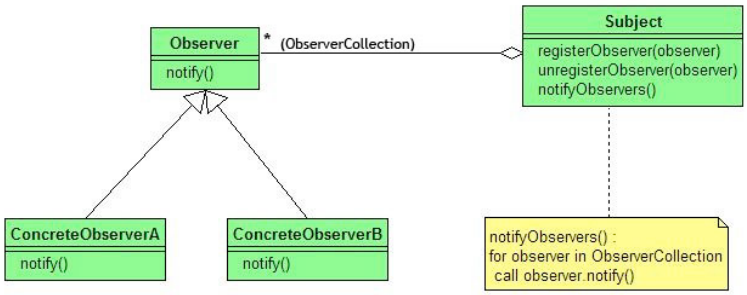
- JavaBean Event Model
- Action Events
- Value Change Events
- Immediate Events
- Phase Events

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Event Handling in Java

- Java Standard Edition 1.1 added an **event model** to Java based on the *Observer* design pattern
- The JavaBean specification adopts this pattern for event handling
- JSF uses the JavaBean event model behind the scenes



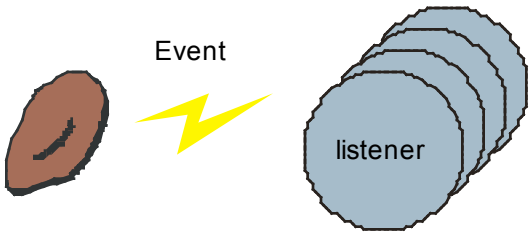
From the Wikipedia:

The observer pattern (a subset of the asynchronous publish/subscribe pattern) is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. It is mainly used to implement distributed event handling systems.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaBean Events

- Events allow JavaBeans to notify listeners of significant occurrences
- The following artifacts comprise an event:
    - An event-object class
    - An event listener interface
    - Register/deregister methods on the bean
    - Firing the event



JavaBean events are almost the opposite of JavaBean methods -- methods let the outside world invoke behavior on the bean, while an event is the bean invoking behavior on "listeners", which are objects that have registered an interest in the event.

Events are especially useful in GUIs, where a user-interface component, say a button, notifies another object when the user clicks the button.

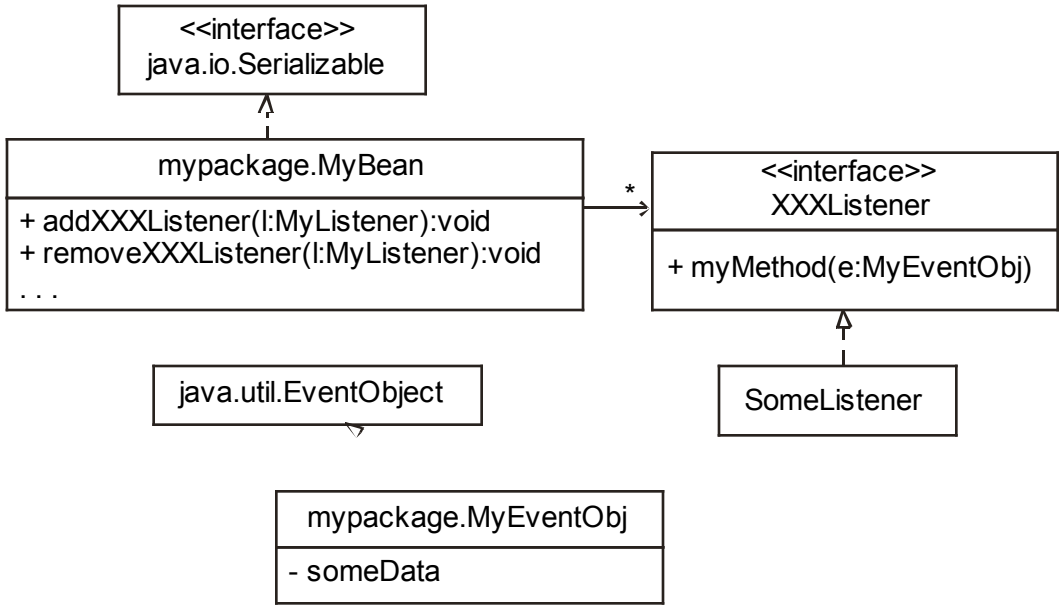
Note that there can be multiple listeners on a given event. The bean notifies all registered listeners when the event occurs.

JavaBean events are a variation on the Observer Design Pattern from the Design Patterns book by Gamma, et. al.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaBean Event Architecture

- The good news is that JSF hides much of this complexity



The idea is that the bean maintains a list of interested listeners, each of which is typed to the interface that we define. The bean provides two methods: one to add a listener to the list, another to remove. Note that the names of these two methods are very significant -- we use introspection to infer the event name from the method names. In other words, whatever you put as the 'XXX' placeholder above will be interpreted as the event name.

We also can define a class that describes the event -- note that we pass an instance of the event class in the interface methods. This is optional -- if you don't have any extra information to pass, you can use the java.util.EventObject class directly.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF Event Categories

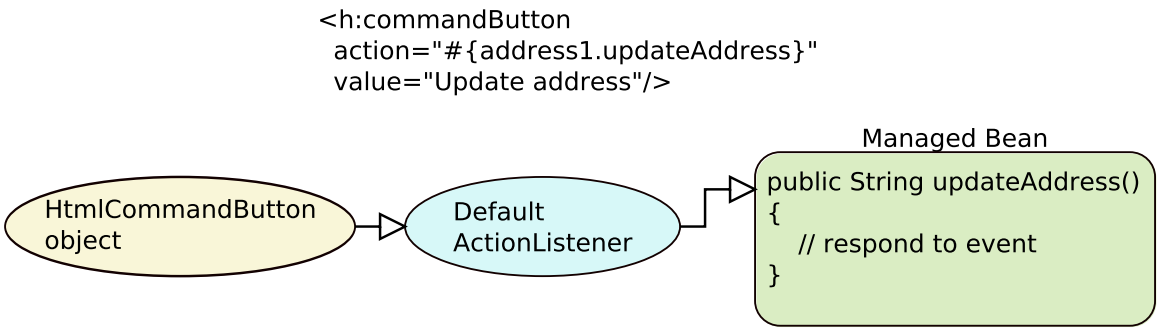
- **Action events** are associated with *commandButtons* and *commandLinks* components
- **Value-change events** are associated with input components such *inputText* and *selectManyMenu*
- **Phase events** are fired during normal operation and can be used in debugging
- **System events** allow finer-grained notification of application and component events

System Events are a new feature in JSF2.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Action Events

- You can configure a JSF-supplied default listener for an action event by writing an **action=** attribute on a commandButton or commandLink tag and writing a **method binding expression** that refers to a managed bean method



It's also possible to use the actionListener attribute instead of the action attribute, and then write a method binding expression that refers to a method with the standard JavaBean event-handling method signature. That form is less commonly used than the example shown here, but is useful if you want to have the same event-handling method be responsible for several components. To use the actionListener attribute, your method's signature should be like:

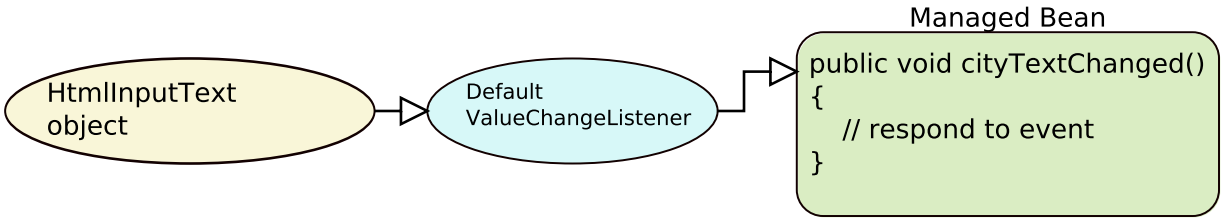
```
public void myActionListener(ActionEvent evt)
{
    System.out.println("In myActionListener");
}
```

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Value Change Events

- **Value change** events fire (at least logically), when the user interacts with a component that has a value (e.g. h:inputText)
- In the absence of JavaScript or Ajax, the event doesn't actually fire until the user submits the page

```
<h:inputText id="city" value="#{address1.city}"
        valueChangeListener="#{address1.cityTextChanged}"/>
```

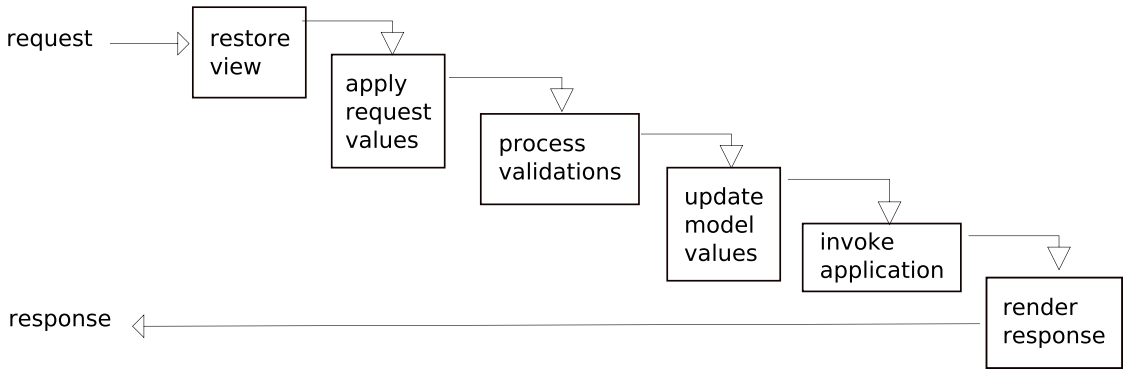


Note that in this case, even if the user types several characters into the h:inputText widget, only one value-change event will fire when the user submits the page.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Immediate vs Non-Immediate Events

- Components detect events during the **Apply Request Value** lifecycle phase, but JSF normally doesn't fire them until the **Invoke Application Phase**
- You can configure an event to be **immediate** which causes JSF to fire the event as part of the **Apply Request Values** phase



You want to use immediate events if you have dependent widgets and are using validation. In other words, you want to handle the event BEFORE any validation occurs.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring an Immediate Event

- Sometimes you want to have dependent widgets, so that when the user changes one widget, the other widget changes in some fashion
- On the source widget's tag:
  - Write an **immediate="true"** attribute
  - Write a JavaScript **onChange** handler that submits the form
  - Write a **valueListener** attribute with a *method binding expression*
- In the event-handling method:
  - Use the passed EventObject to determine the change
  - Use the FacesContext to modify the destination widget as required
  - Force immediate rendering



In the figure, we show a form with a widget for the user to enter a US state or a Canadian province. We want the widget's label to be "State" if the user selects "USA" in the Country list, and "Province" if the user selects Canada.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Immediate Event Example

address.jsp

Address.java

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
2    <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
3    <html>
4      <head>
5        <title>Address Information</title>
6      </head>
7      <f:view>
8        <body>
9          <h4>Enter your address information</h4>
10         <h:form id="addressForm">
11            <p>
12              <h:outputLabel for="street" value="Street:"/>
13              <h:inputText id="street" value="#{address1.street}"
14                required="true"/>
15              <h:message for="street"/>
16            </p>
17            <p>
18              <h:outputLabel for="city" value="City:"/>
19              <h:inputText id="city" value="#{address1.city}"/>
20            </p>
21            <p>
22              <h:outputLabel id="stateLabel" for="state" value="State: "/>
23              <h:inputText id="state" value="#{address1.state}"/>
24            </p>
25            <p>
26              <h:outputLabel for="country" value="Country: "/>
27              <h:selectOneMenu id="country"
28                 value="#{address1.country}"
29                 valueChangeListener="#{address1.countryChanged}"
30                 immediate="true"
31                 onchange="submit()">
32                <f:selectItem itemLabel="USA" itemValue="USA"/>
33                <f:selectItem itemLabel="Canada" itemValue="Canada"/>
34              </h:selectOneMenu>
35            </p>
36            <p>
37              <h:commandButton action="#{address1.updateAddress}"
38                value="Update address"/>
39            </p>
40         </h:form>
41       </body>
42     </f:view>
43   </html>
```

```
1    package info;
2
3    import javax.faces.component.UIViewRoot;
4    import javax.faces.component.html.HtmlOutputLabel;
5    import javax.faces.context.FacesContext;
6    import javax.faces.event.ActionEvent;
7    import javax.faces.event.ValueChangeEvent;
8
9    public class Address
10   {
11       private String street;
12       private String city;
13       private String state;
14       private String country;
15
16       public String getStreet()
17       {
18           return street;
19       }
20
21       public void setStreet(String street)
22       {
23           this.street = street;
24       }
25
26       public String getCity()
27       {
28           return city;
29       }
30
31       public void setCity(String city)
32       {
33           this.city = city;
34       }
35
36       public String getState()
37       {
38           return state;
39       }
40
41       public void setState(String state)
42       {
43           this.state = state;
44       }
45
```

```
46      public String getCountry()
47      {
48          return country;
49      }
50
51      public void setCountry(String country)
52      {
53          this.country = country;
54      }
55
56      public void countryChanged(ValueChangeEvent evt)
57      {
58          FacesContext ctx = FacesContext.getCurrentInstance();
59          UIViewRoot root = ctx.getViewRoot();
60
61          HtmlOutputLabel label =
62              (HtmlOutputLabel)root.findComponent("addressForm:stateLabel");
63
64          String country = (String)evt.getNewValue();
65          System.out.println("In countryChanged, value=" + country);
66
67          String labelText = "State: ";
68          if (country.equals("Canada"))
69              labelText = "Province: ";
70
71          label.setValue(labelText);
72
73          ctx.renderResponse();
74      }
75
76      public String updateAddress()
77      {
78          System.out.println("In updateAddress");
79          return "success";
80      }
81  }
```

Phase Listeners

- The JSF runtime calls any registered **phase listeners** during the request lifecycle
- Phase listeners can be useful during debugging or for advanced uses such as integrating internationalized JSF and Struts applications
- To write a phase listener:
    - Write a class that implements the **PhaseListener** interface
    - Register the class in faces-config.xml

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Phase Listener Example

MyPhaseListener.java

faces-config.xml

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package listeners;
2
3    import javax.faces.event.PhaseEvent;
4    import javax.faces.event.PhaseId;
5    import javax.faces.event.PhaseListener;
6
7    @SuppressWarnings("serial")
8    public class MyPhaseListener implements PhaseListener
9    {
10
11       @Override
12       public void afterPhase(PhaseEvent arg0)
13       {
14           System.out.println("In afterPhase, phaseID=" + arg0.getPhaseId());
15       }
16
17       @Override
18       public void beforePhase(PhaseEvent arg0)
19       {
20           System.out.println("In beforePhase, phaseID=" + arg0.getPhaseId());
21       }
22
23       @Override
24       public PhaseId getPhaseId()
25       {
26           return PhaseId.ANY_PHASE;
27       }
28
29   }
```

```
1    <?xml version="1.0" encoding="UTF-8"?>
2
3    <faces-config
4        xmlns="http://java.sun.com/xml/ns/javaee"
5        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/
7        version="1.2">
8    <managed-bean>
9    <managed-bean-name>address1</managed-bean-name>
10   <managed-bean-class>info.Address</managed-bean-class>
11   <managed-bean-scope>session</managed-bean-scope>
12   </managed-bean>
13   <lifecycle>
14   <phase-listener>listeners.MyPhaseListener</phase-listener>
15   </lifecycle>
16
17   </faces-config>
```

System Events

- **System** events are a new feature of JSF2 that provide finer-grained notification of events
- System events fall into two categories:
    - **Global** events such as application startup and shutdown
    - **Component-specific** events, e.g. *postValidate*

You can register event handlers for these events by writing entries in faces-config.xml.

Also note that you can register component-specific event handlers in a JSF XHTML page using the <f:event> tag.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Global System Events

- **Global** system events let applications listen to events that apply to the whole application

```
1    public class FacesAppListener
2      implements SystemEventListener
3    {
4      public void processEvent(SystemEvent evt)
5         throws AbortProcessingException
6      {
7         if(evt instanceof PostConstructApplicationEvent)
8         {
9            // handle app startup
10        }
11     }
12   }
```

You can use a SystemEventListener in a similar fashion to servlet context listeners, but note that these listeners run AFTER JSF has initialized for the app, while context listeners might run before.

Also note that you must configure these listeners in the faces-config.xml:

```
<application>
  <system-event-listener>
    <system-event-listener-class>
      example.FacesAppListener
  </system-event-listener-class>
  <system-event-class>
    javax.faces.event.PostConstructApplicationEvent
  </system-event-class>
 </system-event-listener>
</application>
```

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About the JSF event handling architecture
- How to write and configure event handlers
- The difference between immediate and non-immediate events

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Panels and Data Tables

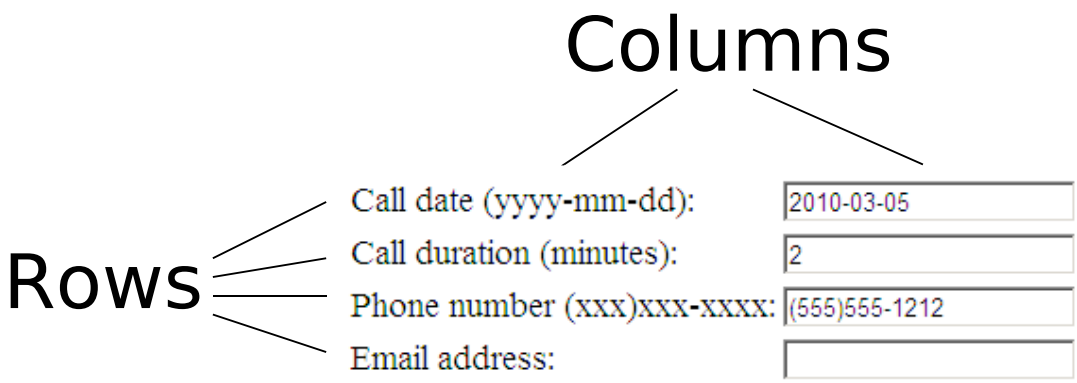
- Panels
- Data Tables

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a PanelGrid?

- The **h:panelGrid** action lets you avoid writing HTML tables by hand
- You can use it to lay out forms by defining JSF components as children of the panel



Panels are essentially a less verbose and less error-prone way to create HTML tables. They are not as powerful as the Data Table component covered later in this chapter.

Using HTML tables to layout forms is considered by some HTML authors to be a bad practice (use CSS positioning instead), but it is widespread, easy and well supported by browsers.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The h:panelGrid Tag

- This action creates an **HtmlPanelGrid** component
- By default, it lays out embedded components in a single column, but you can specify the column count with the **columns=** attribute

```
1    <h:panelGrid columns="2">
2      <h:outputLabel for="calldate"
3         value="Call date (yyyy-mm-dd): "/>
4      <h:inputText id="calldate" value="#{callinfo1.callDate}"
5         required="true"/>
6      <h:outputLabel for="duration"
7         value="Call duration (minutes):"/>
8      <h:inputText id="duration" value="#{callinfo1.callLength}"
9         required="true"/>
10     <h:outputLabel for="phonenumber"
11        value="Phone number (xxx)xxx-xxxx: "/>
12     <h:inputText id="phonenumber" value="#{callinfo1.number}"
13        required="true"/>
14     <h:outputLabel for="email" value="Email address: " />
15     <h:inputText id="email" value="#{callinfo1.email}"/>
16   </h:panelGrid>
```

The HtmlPanelGrid component calculates the number of rows by dividing the number of child components by the column count. In this case, there are eight child components and two columns, yielding four rows.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The h:panelGroup Tag

- This action creates an **HtmlPanelGroup** component that renders its child components as a single entity
- Panel groups are especially useful to group conversion and/or validation error messages in the same table cell as the offending input component

```
1    <h:panelGroup>
2      <h:inputText id="calldate"
3        value="#{callinfo1.callDate}" required="true"/>
4      <h:message for="calldate" style="color:red"/>
5    </h:panelGroup>
```

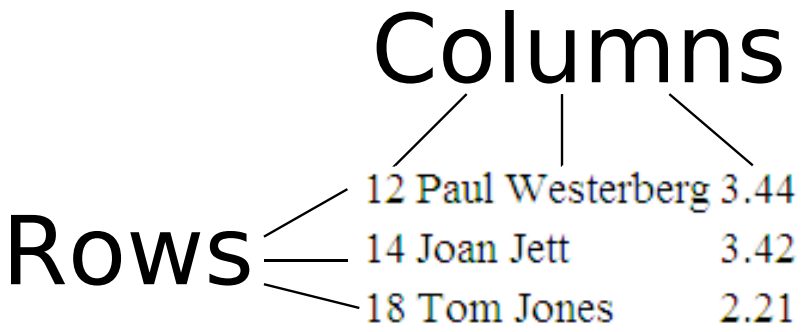


This problem often comes up when a form needs to display validation or conversion messages. Remember that the panel determines rows and columns by counting the child components and that h:message tags conditionally display text. So the layout will change if there's an error message or not and that can throw off the appearance of carefully crafted forms.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Data Tables

- Many Web applications need to display tabular data
- The **h:dataTable** action iterates over a collection and displays a row for each collection element



Data tables are a powerful and flexible component that let JSF applications display tabular data, complete with many styling and appearance options.

You can also customize the table's appearance and behavior by writing a bit of code. For example, if you have a large amount of data to display, you can program a table to display only a portion of the data and then let the user page through the data.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The h:dataTable Tag

- This action creates a **UIData** component
- You use **h:column** child tags to organize the table's columns

simpletable.xhtml

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <html xmlns="http://www.w3.org/1999/xhtml"
2        xmlns:ui="http://java.sun.com/jsf/facelets"
3        xmlns:h="http://java.sun.com/jsf/html"
4        xmlns:f="http://java.sun.com/jsf/core">
5
6      <h:head>
7        <title>Simple Data Table</title>
8      </h:head>
9      <h:body>
10         <h4>Simple Data Table</h4>
11         <h:dataTable value="#{studentBackingBean.students}"
12             var="aStudent">
13           <h:column>
14             <h:outputText value="#{aStudent.studentID}"/>
15           </h:column>
16           <h:column>
17             <h:outputText value="#{aStudent.name}"/>
18           </h:column>
19           <h:column>
20             <h:outputText value="#{aStudent.gpa}"/>
21           </h:column>
22         </h:dataTable>
23       </h:body>
24   </html>
```

Data Table Iteration

- In the **h:dataTable** tag, you write a **value=** attribute, specifying a value binding expression that evaluates to:
  - An array
  - An instance of *java.util.List*
  - An instance of *java.sql.ResultSet*
  - An instance of *javax.servlet.jsp.jstl.sql.Result*
  - An instance of *javax.faces.model.DataModel*
- The table iterates over the collection, naming the current element as specified by the **var=** attribute

```
1    <h:dataTable value="#{studentBackingBean.students}"
2        var="aStudent">
3      <h:column>
4        <h:outputText value="#{aStudent.studentID}"/>
5      </h:column>
6    . . .
```

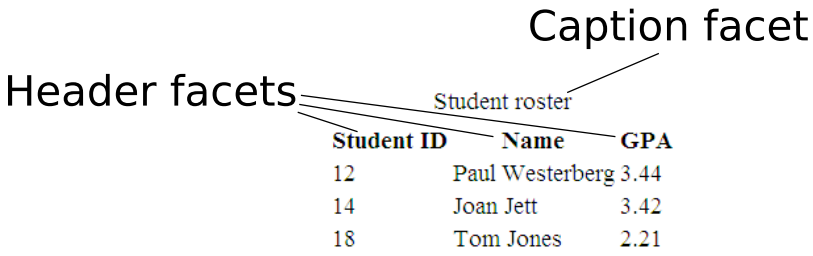
You can also specify a single object in the value= attribute, but that's not very useful: the data table iterates only once.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Data Table Facets

- A **facet** is a child component that has a special purpose for its parent
- Data tables use the following facets:
  - header
  - footer
  - caption

simpletablewithheadings.xhtml



The caption facet is a child of the h:dataTable itself, while you typically use the header facet as a child of a column. Note that the caption header was added to JSF in version 1.2.

A facet can only have a single child component, but you can use a h:panelGroup as the child and within the group, include multiple components.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <html xmlns="http://www.w3.org/1999/xhtml"
2        xmlns:ui="http://java.sun.com/jsf/facelets"
3        xmlns:h="http://java.sun.com/jsf/html"
4        xmlns:f="http://java.sun.com/jsf/core">
5
6      <h:head>
7        <title>Simple Data Table With Headings</title>
8      </h:head>
9      <h:body>
10         <h4>Simple Data Table With Headings</h4>
11         <h:dataTable value="#{studentBackingBean.students}"
12             var="aStudent">
13           <f:facet name="caption">
14             <h:outputText value="Student roster"/>
15           </f:facet>
16           <h:column>
17             <f:facet name="header">
18               <h:outputText value="Student ID"/>
19             </f:facet>
20             <h:outputText value="#{aStudent.studentID}"/>
21           </h:column>
22           <h:column>
23             <f:facet name="header">
24               <h:outputText value="Name"/>
25             </f:facet>
26             <h:outputText value="#{aStudent.name}"/>
27           </h:column>
28           <h:column>
29             <f:facet name="header">
30               <h:outputText value="GPA"/>
31             </f:facet>
32             <h:outputText value="#{aStudent.gpa}"/>
33           </h:column>
34         </h:dataTable>
35       </h:body>
36   </html>
```

Alternating Row Styles

- The **h:dataTable** tag supports a wide range of CSS styling, including alternating row styles

Student ID	Name	GPA
12	Paul Westerberg	3.44
14	Joan Jett	3.42
18	Tom Jones	2.21

alternatingrows.xhtml

The data table component uses the styles passed in the rowClasses= attribute to set style for each row. Not that you're not limited to simply alternating, you could change every fourth row like:

```
. . . rowClasses="evenRow, evenRow, evenRow, oddRow" . . .
```

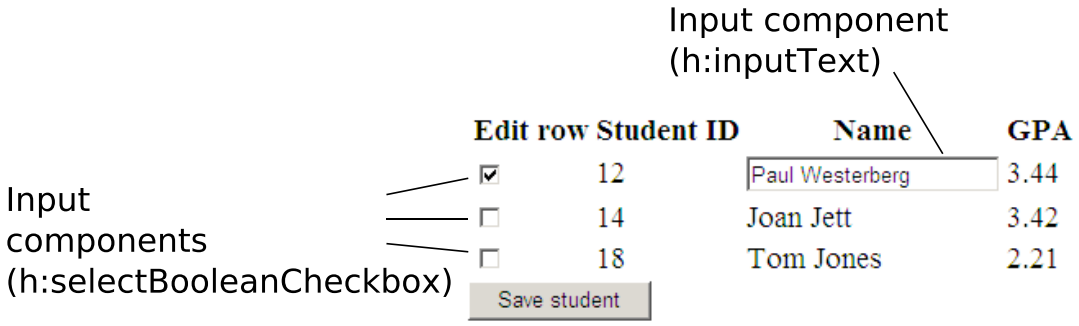
Also note that there's also a similar columnClasses= attribute that lets you style columns separately in the same fashion.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1   <html xmlns="http://www.w3.org/1999/xhtml"
2       xmlns:ui="http://java.sun.com/jsf/facelets"
3       xmlns:h="http://java.sun.com/jsf/html"
4       xmlns:f="http://java.sun.com/jsf/core">
5
6     <h:head>
7       <title>Alternating Rows</title>
8       <style type="text/css">
9         .oddRow{background: #C0C0C0;}
10        .evenRow{background: #DEDEDE;}
11      </style>
12    </h:head>
13    <h:body>
14        <h4>Alternating Rows</h4>
15        <h:dataTable value="#{studentBackingBean.students}"
16            var="aStudent"
17            rowClasses="evenRow,oddRow" >
18          <h:column>
19            <f:facet name="header">
20              <h:outputText value="Student ID"/>
21            </f:facet>
22            <h:outputText value="#{aStudent.studentID}"/>
23          </h:column>
24          <h:column>
25            <f:facet name="header">
26              <h:outputText value="Name"/>
27            </f:facet>
28            <h:outputText value="#{aStudent.name}"
29              rendered="#{not aStudent.editable}"/>
30            <h:inputText value="#{aStudent.name}"
31              rendered="#{aStudent.editable}"/>
32          </h:column>
33          <h:column>
34            <f:facet name="header">
35              <h:outputText value="GPA"/>
36            </f:facet>
37            <h:outputText value="#{aStudent.gpa}"/>
38          </h:column>
39        </h:dataTable>
40      </h:body>
41  </html>
```

Components in Data Table Cells

- The previous examples have shown using **outputText** components in the table cells, but you are not limited to that
- You can use outputLinks, commandButtons, inputText or any other JSF component



Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Editable Table Cells

- By using conditional rendering of input and output components, you can create tables that let the user modify data in the table
- In this technique, you render an input component if the user marks a row as "editable" and an output component if not

editabletable.xhtml

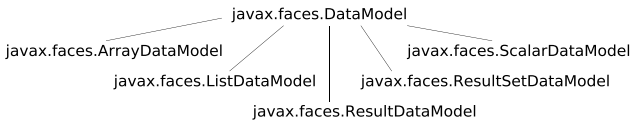
Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <html xmlns="http://www.w3.org/1999/xhtml"
2        xmlns:ui="http://java.sun.com/jsf/facelets"
3        xmlns:h="http://java.sun.com/jsf/html"
4        xmlns:f="http://java.sun.com/jsf/core">
5
6      <h:head>
7        <title>Editable Data Table</title>
8      </h:head>
9      <h:body>
10         <h4>Editable Data Table</h4>
11         <h:form>
12             <h:dataTable value="#{studentBackingBean.students}"
13                 var="aStudent">
14               <h:column>
15                 <f:facet name="header">
16                   <h:outputText value="Edit row"/>
17                 </f:facet>
18                 <h:selectBooleanCheckbox
19                   value="#{aStudent.editable}"
20                   onclick="submit()"/>
21               </h:column>
22               <h:column>
23                 <f:facet name="header">
24                   <h:outputText value="Student ID"/>
25                 </f:facet>
26                 <h:outputText value="#{aStudent.studentID}"/>
27               </h:column>
28               <h:column>
29                 <f:facet name="header">
30                   <h:outputText value="Name"/>
31                 </f:facet>
32                 <h:outputText value="#{aStudent.name}"
33                   rendered="#{not aStudent.editable}"/>
34                 <h:inputText value="#{aStudent.name}"
35                   rendered="#{aStudent.editable}"/>
36               </h:column>
37               <h:column>
38                 <f:facet name="header">
39                   <h:outputText value="GPA"/>
40                 </f:facet>
41                 <h:outputText value="#{aStudent.gpa}"/>
42               </h:column>
43             </h:dataTable>
44             <h:commandButton action="#{studentBackingBean.save}"
45               value="Save student"/>
```

```
46          </h:form>
47        </h:body>
48    </html>
```

Table Data Models

- JSF defines the **javax.faces.model.DataModel** class and subclasses that represent sources of data
- The **h:dataTable** action accepts an **value=** attribute in which you can specify a data source
- If you specify anything EXCEPT a DataModel, the action wraps the source with a DataModel
- You CAN specify a DataModel directly, and this can be a useful technique



javax.faces.model.DataModel
addDataModelListener(dml:DataModelListener):void getDataModelListeners():DataModelListener[] getRowCount():int getRowData():Object getRowIndex():int getWrappedData():Object isRowAvailable():boolean removeDataModelListener(dml:DataModelListener):void setRowIndex(index:int):void setWrappedData(data:Object):void

The UIData component actually never interacts directly with the collection. Instead, it always accesses the collection through the data model object.

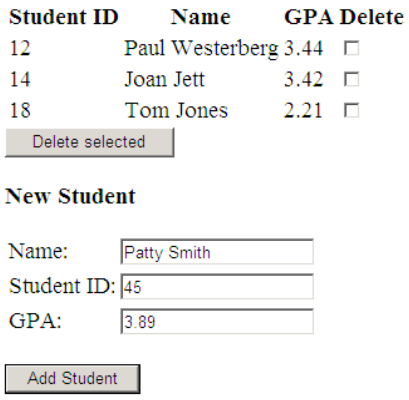
Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a Data Model

- The benefit of directly providing a DataModel is that you can then dynamically change the collection
- For example, you can let the user add or delete table rows

datamodel.xhtml

StudentBackingBeanWithDataModel.java



Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <html xmlns="http://www.w3.org/1999/xhtml"
2        xmlns:ui="http://java.sun.com/jsf/facelets"
3        xmlns:h="http://java.sun.com/jsf/html"
4        xmlns:f="http://java.sun.com/jsf/core">
5
6      <h:head>
7        <title>Dynamic Data Table</title>
8      </h:head>
9      <h:body>
10         <h4>Dynamic Data Table</h4>
11         <h:form>
12             <h:dataTable
13                 value="#{studentBackingBeanWithDataModel.students}"
14                 var="aStudent">
15               <h:column>
16                 <f:facet name="header">
17                   <h:outputText value="Student ID"/>
18                 </f:facet>
19                 <h:outputText value="#{aStudent.studentID}"/>
20               </h:column>
21               <h:column>
22                 <f:facet name="header">
23                   <h:outputText value="Name"/>
24                 </f:facet>
25                 <h:outputText value="#{aStudent.name}" />
26               </h:column>
27               <h:column>
28                 <f:facet name="header">
29                   <h:outputText value="GPA"/>
30                 </f:facet>
31                 <h:outputText value="#{aStudent.gpa}"/>
32               </h:column>
33               <h:column>
34                 <f:facet name="header">
35                   <h:outputText value="Delete"/>
36                 </f:facet>
37                 <h:selectBooleanCheckbox
38                   value="#{aStudent.editable}"
39                   onclick="submit()"/>
40               </h:column>
41             </h:dataTable>
42             <h:commandButton
43               action="#{studentBackingBeanWithDataModel.deleteSelected}"
44               value="Delete selected"/>
45         </h:form>
```

```
46          <h4>New Student</h4>
47          <h:form>
48            <h:panelGrid columns="2">
49              <h:outputLabel for="name"
50                value="Name: "/>
51              <h:panelGroup>
52                <h:inputText id="name"
53                   value="#{student.name}"
54                   required="true"/>
55                <h:message for="name" style="color:red"/>
56              </h:panelGroup>
57              <h:outputLabel for="studentID"
58                value="Student ID:"/>
59              <h:panelGroup>
60                <h:inputText id="studentID"
61                    value="#{student.studentID}"
62                    required="true"/>
63                <h:message for="studentID" style="color:red"/>
64              </h:panelGroup>
65              <h:outputLabel for="gpa"
66                value="GPA: "/>
67              <h:panelGroup>
68                <h:inputText id="gpa"
69                   value="#{student.gpa}"
70                   required="true"/>
71                <h:message for="gpa" style="color:red"/>
72              </h:panelGroup>
73            </h:panelGrid>
74            <p>
75              <h:commandButton value="Add Student"
76                action="#{studentBackingBeanWithDataModel.addStudent}"/>
77            </p>
78          </h:form>
79        </h:body>
80    </html>
```

```
1    package university;
2
3    import java.util.ArrayList;
4    import java.util.List;
5
6    import javax.el.ELContext;
7    import javax.el.ExpressionFactory;
8    import javax.el.ValueExpression;
9    import javax.faces.application.Application;
10   import javax.faces.bean.ManagedBean;
11   import javax.faces.bean.SessionScoped;
12   import javax.faces.context.FacesContext;
13   import javax.faces.model.DataModel;
14   import javax.faces.model.ListDataModel;
15
16   @ManagedBean(name="studentBackingBeanWithDataModel")
17   @SessionScoped
18   public class StudentBackingBeanWithDataModel
19   {
20       private DataModel model;
21
22       public DataModel getStudents()
23       {
24           return model;
25       }
26
27       public StudentBackingBeanWithDataModel()
28       {
29           List<Student> students = new ArrayList<Student>();
30           students.add(new Student("Paul Westerberg", 3.44, 12));
31           students.add(new Student("Joan Jett", 3.42, 14));
32           students.add(new Student("Tom Jones", 2.21, 18));
33           model = new ListDataModel(students);
34       }
35
36       public String deleteSelected()
37       {
38           System.out.println("In deleteSelected");
39
40           List<Student> oldList = (List<Student>) model.getWrappedData();
41           List<Student> newList = new ArrayList<Student>();
42
43           for (Student aStudent : oldList)
44           {
45               if (!aStudent.isEditable())
```

```
46                  newList.add(aStudent);
47          }
48
49          model.setWrappedData(newList);
50          return null;
51      }
52
53      public String addStudent()
54      {
55          FacesContext ctx = FacesContext.getCurrentInstance();
56          Application app = ctx.getApplication();
57          ExpressionFactory fact = app.getExpressionFactory();
58          ELContext elContext = ctx.getELContext();
59          ValueExpression valExpr = fact.createValueExpression(elContext,
60                  "#{student}", Student.class);
61          Student student = (Student)valExpr.getValue(elContext);
62
63          System.out.println("In addStudent: " + student);
64
65          List<Student> oldList = (List<Student>) model.getWrappedData();
66          List<Student> newList = new ArrayList<Student>(oldList);
67          newList.add(student);
68          model.setWrappedData(newList);
69          return null;
70      }
71  }
```

Advanced Data Table Topics

The remaining part of this chapter covers how to use the JSF data table component to display rows and columns from a relational database

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a Data Table with a Database

- A common requirement is to use a data tables to display/modify data from a database
- There are two basic strategies:
    - Use a managed bean that queries a database table and simply exposes a JDBC or JSTL **ResultSet** to a data table component
    - Use a managed bean that queries a database table and exposes a collection of Java **domain objects**

```
CREATE TABLE student (
  studentid INTEGER NOT NULL,
  name VARCHAR(50),
  gpa DOUBLE,
  PRIMARY KEY(studentid))
```

Student ID	Name	GPA
3	John Smith	3.12
4	Sue Harrison	3.45
5	Harry Wolfe	2.33
6	Kevin McAlpine	1.66
7	Mary McDonald	3.15
8	Virginia Johnson	2.56
9	Stan Mossberg	3.58
10	William Carter	3.44
11	Pam Caten	2.77
12	Emma Sanders	4.55
13	Gail Sayton	4.55
14	Sasha Wilmead	4.55

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Accessing the Database

- There are several common techniques to access a database in Java:
    - JDBC
    - The DAO pattern (roll your own)
    - JDBC framework such as Spring DAO
    - Object/relational mapping framework such as Hibernate or JPA

If you decide to simply expose a ResultSet to the data table, then you can use JDBC to execute the query and obtain the result set.

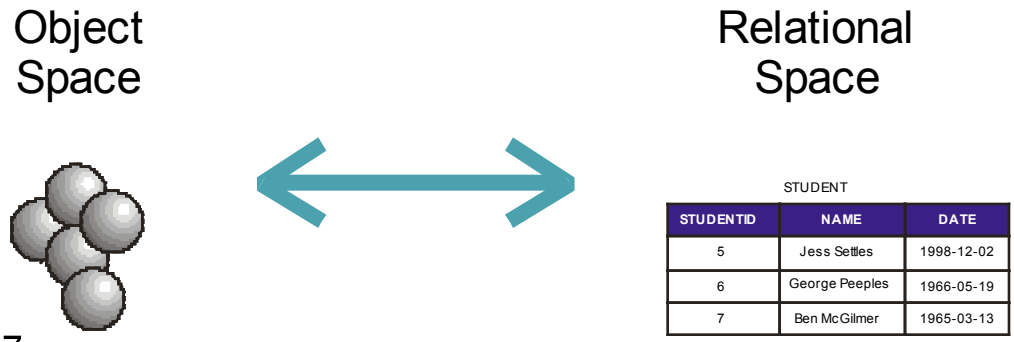
If you decide to expose "domain objects", then you can use JDBC, but it's probably easier to use one of the frameworks shown here.

In the rest of this chapter, we'll explore the JPA approach.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is JPA?

- **Java Persistence** is the official Sun API for object/relational mapping
- Originally only for use with EJB3, Java Persistence is now a separate API and can be used in Java SE applications
- JPA is a required part of Java Enterprise Edition, Version 5



Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Java Persistence Ancestry

- Java Persistence derives from EJB2, Hibernate, TopLink and Java Data Objects (JDO)
- The thought was to use the best features from existing O/R frameworks and create a single API "to rule them all"

As is often the case in Java, there are several competing initiatives for object-relational mapping, resulting in a lot of "reinventing the wheel".

When the EJB3 specification was in progress, the committee decided to attempt to unify the various projects and create a specification that was acceptable to all.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Java Persistence Overview

- Write **entity** classes
- Obtain an **EntityManager** reference so you can work with the persistent objects
- Configure Java Persistence by writing an XML file

An entity is a Java class with annotations that map data to columns in a database.

The EntityManager lets you work with entities, including executing queries and saving to the database.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JPA Data Table Example

Student.java

persistence.xml

simpletablejpa.jsp

StudentBackingBean.java

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package student;
2
3    import javax.persistence.Entity;
4    import javax.persistence.Id;
5
6    @Entity
7    public class Student
8    {
9        private String name;
10       private double gpa;
11       @Id
12       private int studentID;
13
14       public Student(String name, double gpa, int studentID)
15       {
16           this.name = name;
17           this.gpa = gpa;
18           this.studentID = studentID;
19       }
20
21       public Student()
22       {
23
24       }
25
26       public String getName()
27       {
28           return name;
29       }
30
31       public void setName(String name)
32       {
33           this.name = name;
34       }
35
36       public double getGpa()
37       {
38           return gpa;
39       }
40
41       public void setGpa(double gpa)
42       {
43           this.gpa = gpa;
44       }
45
```

```
46      public int getStudentID()
47      {
48          return studentID;
49      }
50
51      public void setStudentID(int studentID)
52      {
53          this.studentID = studentID;
54      }
55
56      @Override
57      public String toString()
58      {
59          return "StudentID: " + studentID +
60              " Name: " + name +
61              " GPA: " + gpa;
62      }
63  }
```

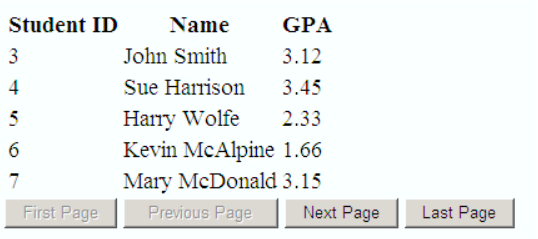
```
1    <?xml version="1.0" encoding="UTF-8"?>
2    <persistence version="1.0"
3        xmlns="http://java.sun.com/xml/ns/persistence"
4        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
6        http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
7        <persistence-unit name="default"
8           transaction-type="RESOURCE_LOCAL">
9            <provider>
10             org.eclipse.persistence.jpa.PersistenceProvider
11           </provider>
12           <class>student.Student</class>
13           <properties>
14               <property name="eclipselink.target-server"
15                 value="WebLogic_10" />
16               <property name="eclipselink.logging.level"
17                 value="WARNING" />
18               <property name="eclipselink.jdbc.driver"
19                 value="org.hsqldb.jdbcDriver" />
20               <property name="eclipselink.jdbc.platform"
21                 value="org.eclipse.persistence.platform.database.oracle.OraclePlatfo
22               <property name="eclipselink.jdbc.url"
23                 value="jdbc:hsqldb:hsql://localhost" />
24               <property name="eclipselink.jdbc.user" value="sa" />
25               <property name="eclipselink.jdbc.password" value="" />
26           </properties>
27       </persistence-unit>
28   </persistence>
```

```
1    <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
2    <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
3    <html>
4      <head>
5        <title>Simple Data Table With Headings</title>
6      </head>
7      <f:view>
8        <body>
9          <h4>Simple Data Table With Headings</h4>
10         <h:dataTable value="#{studentBackingBean.students}"
11             var="aStudent">
12           <h:column>
13             <f:facet name="header">
14               <h:outputText value="Student ID"/>
15             </f:facet>
16             <h:outputText value="#{aStudent.studentID}"/>
17           </h:column>
18           <h:column>
19             <f:facet name="header">
20               <h:outputText value="Name"/>
21             </f:facet>
22             <h:outputText value="#{aStudent.name}"/>
23           </h:column>
24           <h:column>
25             <f:facet name="header">
26               <h:outputText value="GPA"/>
27             </f:facet>
28             <h:outputText value="#{aStudent.gpa}"/>
29           </h:column>
30         </h:dataTable>
31       </body>
32     </f:view>
33   </html>
```

```
1    package student;
2
3    import java.util.List;
4
5    import javax.persistence.EntityManager;
6    import javax.persistence.EntityManagerFactory;
7    import javax.persistence.Persistence;
8    import javax.persistence.Query;
9
10   public class StudentBackingBean
11   {
12
13       public List<Student> getStudents()
14       {
15           EntityManagerFactory emf =
16               Persistence.createEntityManagerFactory("default");
17           EntityManager entityManager =
18               (EntityManager) emf.createEntityManager();
19           entityManager.getTransaction().begin();
20
21           Query q = entityManager.createQuery(
22                   "SELECT s FROM Student s");
23           List<Student> students = q.getResultList();
24
25           entityManager.getTransaction().commit();
26           entityManager.close();
27           emf.close();
28
29           return students;
30       }
31   }
```

Paging Large Result Sets

- For large databases, it's not a good idea to display all of the rows at once, so JPA supports **paged** queries
- You can use a data table and a few buttons to let the user page through such large result sets



Some such applications use "arrow" icons instead of text-based buttons as shown here.

Note how the "First Page" and "Previous Page" buttons are grayed-out because the table is displaying the first page.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JPA Paging Support

- After you create a JPA query, you can set the **first result** and **maximum results**
- A backing bean can maintain these values based on user interaction

```
1    Query q = entityManager.createQuery(
2      "SELECT s FROM Student s");
3    q.setFirstResult(curIndex);
4    q.setMaxResults(rowsPerPage);
5
6    List<Student> students = q.getResultList();
```


Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Enabling and Disabling Paging Buttons

- With some creative JSF expressions, you can enable and disable the paging buttons as required

```
1     <h:commandButton
2       action="#{studentBackingBean.firstPage}"
3       value="First Page"
4       disabled="#{studentBackingBean.curIndex == 0}"/>
5     <h:commandButton
6       action="#{studentBackingBean.previousPage}"
7       value="Previous Page"
8       disabled="#{studentBackingBean.curIndex == 0}"/>
9     . . .
```

The "Next Page" and "Last Page" buttons are not shown in this abbreviated listing, but are shown in the complete listing on a later page.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Paging Example

pagedtablejpa.jsp

StudentBackingBeanPaged.java

The Student.java and persistence.xml files for this application are the same as for the last one, so they are not shown here.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
2    <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
3    <html>
4      <head>
5        <title>Paged Data Table With Headings</title>
6      </head>
7      <f:view>
8        <body>
9          <h4>Paged Data Table With Headings</h4>
10         <h:form>
11             <h:dataTable value="#{studentBackingBean.students}"
12                 var="aStudent">
13               <h:column>
14                 <f:facet name="header">
15                   <h:outputText value="Student ID"/>
16                 </f:facet>
17                 <h:outputText value="#{aStudent.studentID}"/>
18               </h:column>
19               <h:column>
20                 <f:facet name="header">
21                   <h:outputText value="Name"/>
22                 </f:facet>
23                 <h:outputText value="#{aStudent.name}"/>
24               </h:column>
25               <h:column>
26                 <f:facet name="header">
27                   <h:outputText value="GPA"/>
28                 </f:facet>
29                 <h:outputText value="#{aStudent.gpa}"/>
30               </h:column>
31             </h:dataTable>
32            <h:commandButton
33               action="#{studentBackingBean.firstPage}"
34               value="First Page"
35               disabled="#{studentBackingBean.curIndex == 0}"/>
36            <f:verbatim> </f:verbatim>
37            <h:commandButton
38               action="#{studentBackingBean.previousPage}"
39               value="Previous Page"
40               disabled="#{studentBackingBean.curIndex == 0}"/>
41            <f:verbatim> </f:verbatim>
42            <h:commandButton
43               action="#{studentBackingBean.nextPage}"
44               value="Next Page"
45               disabled=
```

```
46                "#{studentBackingBean.curIndex + studentBackingBean.rowsPerPage >=
47                    studentBackingBean.studentCount}"/>
48            <f:verbatim> </f:verbatim>
49            <h:commandButton
50              action="#{studentBackingBean.lastPage}"
51              value="Last Page"
52              disabled=
53              "#{studentBackingBean.curIndex + studentBackingBean.rowsPerPage >=
54                  studentBackingBean.studentCount}"/>
55          </h:form>
56       </body>
57     </f:view>
58   </html>
```

```
1    package student;
2
3    import java.util.List;
4
5    import javax.persistence.EntityManager;
6    import javax.persistence.EntityManagerFactory;
7    import javax.persistence.Persistence;
8    import javax.persistence.Query;
9
10   public class StudentBackingBeanPaged
11   {
12       private int rowsPerPage = 5;
13       private int curIndex = 0;
14
15       public List<Student> getStudents()
16       {
17           EntityManagerFactory emf = Persistence
18                   .createEntityManagerFactory("default");
19           EntityManager entityManager = (EntityManager) emf.createEntityManager();
20           entityManager.getTransaction().begin();
21
22           Query q = entityManager.createQuery("SELECT s FROM Student s");
23           q.setFirstResult(curIndex);
24           q.setMaxResults(rowsPerPage);
25
26           List<Student> students = q.getResultList();
27
28           entityManager.getTransaction().commit();
29           entityManager.close();
30           emf.close();
31
32           return students;
33       }
34
35       public String nextPage()
36       {
37           if (curIndex + rowsPerPage <= getStudentCount())
38           {
39               curIndex += rowsPerPage;
40           }
41           return null;
42       }
43
44       public String previousPage()
45       {
```

```
46          if (curIndex > 0)
47          {
48              if (curIndex - rowsPerPage >= 0)
49                  curIndex -= rowsPerPage;
50              else
51                  curIndex = 0;
52          }
53          return null;
54      }
55
56      public String firstPage()
57      {
58          curIndex = 0;
59          return null;
60      }
61
62      public String lastPage()
63      {
64          curIndex = getStudentCount() - rowsPerPage;
65          return null;
66      }
67
68      public int getStudentCount()
69      {
70          EntityManagerFactory emf = Persistence
71                  .createEntityManagerFactory("default");
72          EntityManager entityManager = (EntityManager) emf.createEntityManager();
73          entityManager.getTransaction().begin();
74
75          String oql = "SELECT COUNT(s.studentID) FROM Student s";
76          Query q = entityManager.createQuery(oql);
77
78          Number countResult = (Number) q.getSingleResult();
79          entityManager.getTransaction().commit();
80          entityManager.close();
81          emf.close();
82
83          return countResult.intValue();
84      }
85
86      public int getRowsPerPage()
87      {
88          return rowsPerPage;
89      }
90
```

```
91        public void setRowsPerPage(int rowCount)
92        {
93            this.rowsPerPage = rowCount;
94        }
95
96        public int getCurIndex()
97        {
98            return curIndex;
99        }
100
101       public void setCurIndex(int curIndex)
102       {
103           this.curIndex = curIndex;
104       }
105   }
```

Chapter Summary

In this chapter, you learned:

- How to use panels to lay out forms and other JSF components
- How to use data tables to display tabular data

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF and Internationalization

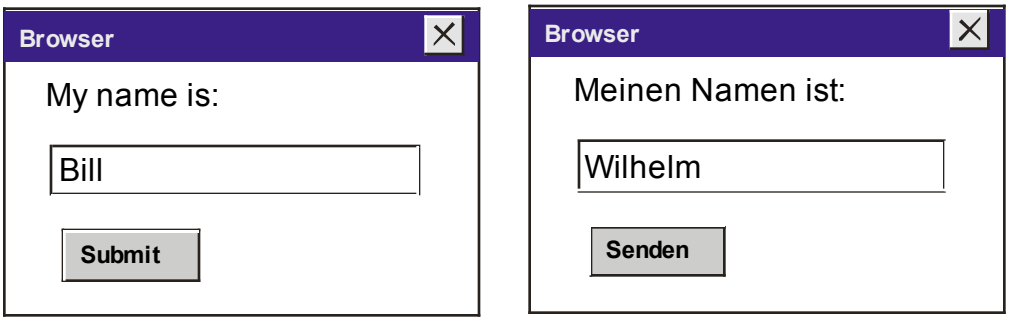
- Language and Country Codes
- Message Bundles
- Formatting Dates and Numbers

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Internationalization ?

- Internationalization, often abbreviated as I18N, is the act of writing a program so that it can adapt to different national languages and cultures
- Localization (L10N), is the act of actually adapting the program to a particular locale



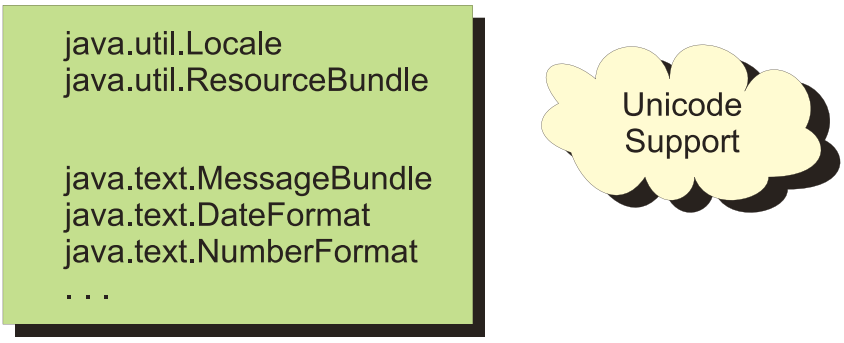
It's a cliche, but the world really is getting smaller. It's likely that JSF applications will be used by people who do not use English as their everday language. Luckily, JSF makes it fairly easy to adapt your program for users in other locales.

Note that localization means more than just translating messages -- it also means using local customs for dates, times and currency.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSE Support for I18N

- The base Java platform provides considerable support for I18N
- JSF builds on and extends that support



Java itself was designed from the very beginning to support I18N. The util and text packages especially provide a wealth of function. The basic class is the Locale class, which gathers together all of the information about a given locale. The ResourceBundle class provides a mechanism to support translatable strings (we've used it indirectly with the ApplicationResources.properties file). Finally, the text package contains several classes that let you format dates, currency and so forth in a locale-specific fashion.

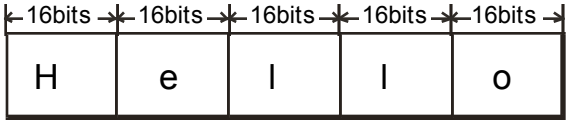
Though the basic Java support is not that difficult to use, the good news is that JSF handles most of this for you!

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

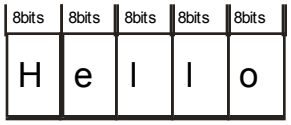
What is Unicode?

- Unicode is an industry standard for representing characters from all of the world's character sets
- Unicode replaces older standards such as ASCII and EBCDIC
- Unicode uses 8, 16, 32 bits or more for each character

UTF-16



Windows-1252



Windows-1252 is a text encoding that's (obviously) the default on Windows. It essentially is a fixed-length extension to ASCII.

UTF-8 and UTF-16 are Unicode text encodings that use a variable-length character. In UTF-16 (shown here), each character takes up at least 16 bits, but if a given character is outside of the 65,536 "default" characters, the size expands, perhaps to 64 bits. UTF-8 uses a minimum 8-bit character size, but has a much smaller "default" range, only 256 characters.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Language and Country Codes

- The International Organization for Standards (ISO) has defined codes for both languages and countries
- The combination of a language code and a country code specifies both a national language and local customs (e.g. How to format dates and currency)

```
Language Codes
--------------
en      English
de      German
fr      French
es      Spanish
ru      Russian
. . .
```

```
Country Codes
--------------
US   United States
GB   United Kingdom
CA   Canada
. . .
```

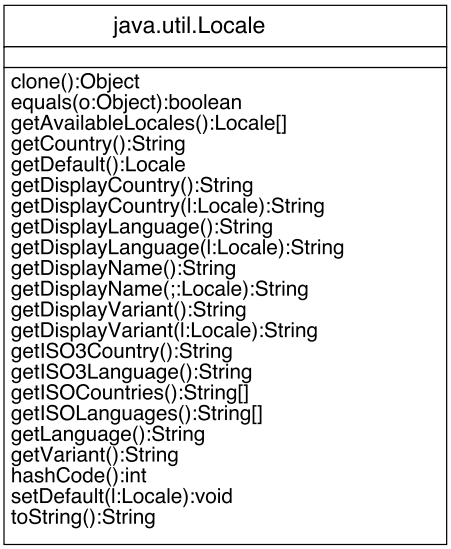
Java uses ISO's codes as a way to categorize resource bundles containing translatable strings. For more information on the ISO codes, see :

`http://www.iso.org/iso/en/prods-services/popstds/popstdsindex.html.`

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Locale?

- Instances of the **java.util.Locale** class represent geographical regions
- JSF will automatically create a Locale based on the HTTP request but the application can explicitly specify one, perhaps based on a user's selection



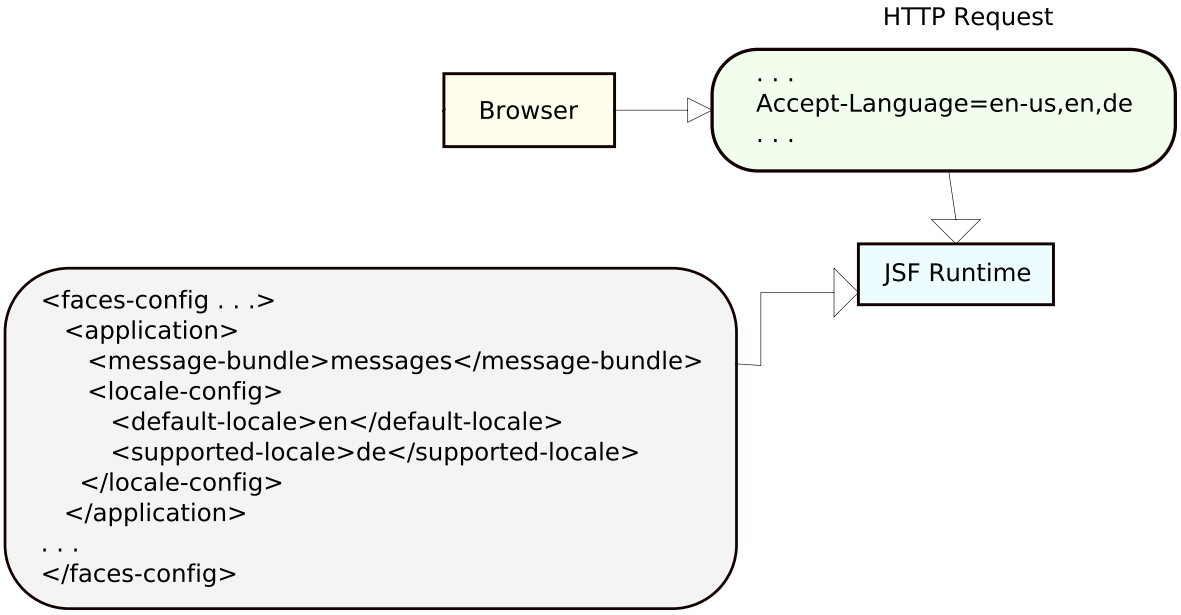
You can create a Locale by specifying a language or a language and a country code:

```
Locale localeEn = new Locale("en");
Locale localeEnUs = new Locale("en", "US");
```

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

How Does JSF Determine Locale?

- The JSF runtime determines locale by matching the **Accept-Language** HTTP header to an entry in faces-config.xml



The JSF runtime treats the Accept-Language header value as a priority ordered list. It attempts to match the first value to a "supported-locale" element in faces-config.xml.

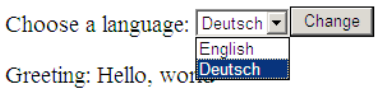
If there's no match, it tries again with the next value until a match is found. If there is no match after going through all of the request-header values, JSF chooses the "default-locale" specified in faces-config.xml.

Note that this establishes a locale for the application, but it's certainly possible for the application to choose a different locale, perhaps under user control.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Selecting a Locale Explicitly

- Many I18N applications use the automatic locale selection as a starting point, but also let the user explicitly choose a locale, or at least a language




```
1    FacesContext ctx =
2       FacesContext.getCurrentInstance();
3    UIViewRoot root = ctx.getViewRoot();
4
5    Locale locale = . . . ;
6    root.setLocale(locale);
```

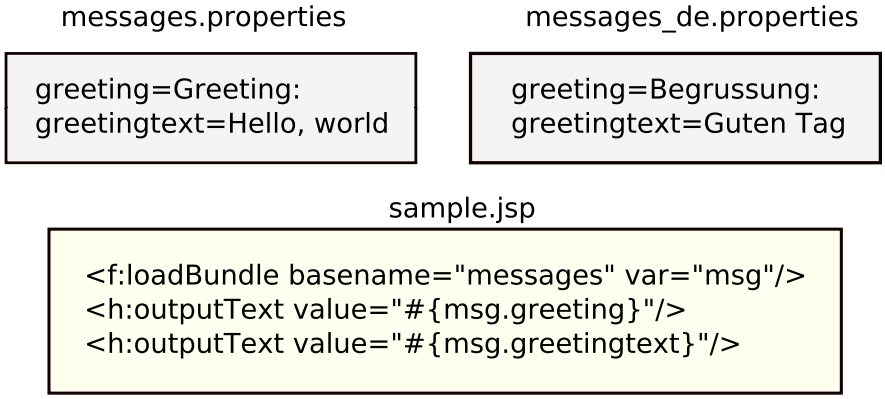
The code shown here is from the managed-bean method invoked by the "Change" button.

Note that you can change the locale by calling setLocale on the UIViewRoot component, which is the parent of all components on a page (it's created by the f:view action).

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Message Bundle Property Files

- Internationalized applications should not hard-code displayable text
- Instead, define such text in a message bundle and specify the bundle's **base name** with the **f:loadBundle** action
- JSF will select the appropriate message bundle file based on the current locale and the bundle's base name



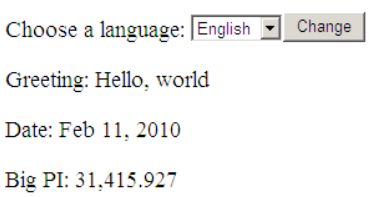
The trick here is in how you name the resource-bundle files. The "base" file contains all of the translatable strings for the server's default locale. Then you create additional files for each of the locales you wish to support, adding the language code and optionally the country code to the file name. Then at runtime, the system will choose the appropriate file based on the locale.

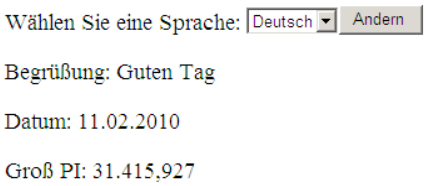
Note that validation and conversion strings are automatically read from a message bundle, so you don't need to do anything special for them.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Formatting Dates and Numbers

- Different cultures expect dates and numbers to be formatted differently
- The **f:convertDateTime** and **f:convertNumber** actions use Java's **DateFormat** and **NumberFormat** classes to convert in a locale-sensitive fashion





Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete I18N Example

faces-config.xml

messages.properties

messages_de.properties

sample.jsp

MyBean.java

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <?xml version="1.0" encoding="UTF-8"?>
2
3    <faces-config
4        xmlns="http://java.sun.com/xml/ns/javaee"
5        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/
7        version="1.2">
8        <application>
9            <message-bundle>messages</message-bundle>
10           <locale-config>
11               <default-locale>en</default-locale>
12               <supported-locale>de</supported-locale>
13           </locale-config>
14       </application>
15       <managed-bean>
16           <managed-bean-name>myBean</managed-bean-name>
17           <managed-bean-class>mypackage.MyBean</managed-bean-class>
18           <managed-bean-scope>session</managed-bean-scope>
19       </managed-bean>
20
21   </faces-config>
```

```
1    title=I18N Application
2    greeting=Greeting:
3    greetingtext=Hello, world
4    date=Date:
5    bigpi=Big PI:
6    language=Choose a language:
7    update=Change
```

```
1    title=I18N Programm
2    greeting=Begrussflung:
3    greetingtext=Guten Tag
4    date=Datum:
5    bigpi=Gross PI:
6    language=Wahlen Sie eine Sprache:
7    update=Andern
```

```
1    <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
2    <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
3    <f:view>
4      <html>
5        <head>
6          <title><h:outputText value="#{msg.title}"/></title>
7        </head>
8        <body>
9          <h4><h:outputText value="#{msg.title}"/></h4>
10         <h:form>
11           <f:loadBundle basename="messages" var="msg"/>
12           <p>
13             <h:outputLabel for="languages"
14                value="#{msg.language}" />
15             <h:selectOneMenu id="languages"
16               value="#{myBean.language}">
17               <f:selectItem itemLabel="English" itemValue="en"/>
18               <f:selectItem itemLabel="Deutsch" itemValue="de"/>
19             </h:selectOneMenu>
20             <h:commandButton action="#{myBean.update}"
21               value="#{msg.update}" />
22           </p>
23           <p>
24              <h:outputText value="#{msg.greeting}"/>
25              <h:outputText value="#{msg.greetingtext}"/>
26           </p>
27           <p>
28              <h:outputText value="#{msg.date}"/>
29              <h:outputText value="#{myBean.date}">
30                <f:convertDateTime dateStyle="medium"/>
31              </h:outputText>
32           </p>
33           <p>
34              <h:outputText value="#{msg.bigpi}"/>
35              <h:outputText value="#{myBean.bigPi}">
36                <f:convertNumber />
37              </h:outputText>
38           </p>
39         </h:form>
40         <hr>
41         <h:outputLink value="clearsession.jsp" >
42           Clear session
43         </h:outputLink>
44       </body>
45     </html>
```

```
46   </f:view>
```

```
1    package mypackage;
2
3    import java.util.Date;
4    import java.util.Locale;
5
6    import javax.faces.component.UIViewRoot;
7    import javax.faces.context.FacesContext;
8
9    public class MyBean
10   {
11       private Locale locale;
12
13       public String getLanguage()
14       {
15           if (locale == null)
16           {
17               FacesContext ctx = FacesContext.getCurrentInstance();
18               UIViewRoot root = ctx.getViewRoot();
19               locale = root.getLocale();
20           }
21
22           return locale.getLanguage();
23       }
24
25       public void setLanguage(String language)
26       {
27           locale = new Locale(language);
28       }
29
30       public Date getDate()
31       {
32           return new Date();
33       }
34
35       public double getBigPi()
36       {
37           return Math.PI * 10000;
38       }
39
40       public String update()
41       {
42           FacesContext ctx = FacesContext.getCurrentInstance();
43           UIViewRoot root = ctx.getViewRoot();
44           root.setLocale(locale);
45           return null;
```

```
46          }
47      }
```

Chapter Summary

In this chapter, you learned:

- The basics of internationalization
- How JSF determines locales
- How to write a JSF application that supports multiple locales

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Facelets

- What is Facelets?
- A Simple Facelets Application

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Facelets?

- Facelets was originally designed as third-party extension to JSF, but is standard starting with JSF2
- Facelets lets you write JSF pages as XHTML files rather than JSPs
- Facelets provides a templating technique where you can define a common look, feel and layout and then base all of your pages on this template - that way if you need to change the layout, you don't need to modify all of the pages, just the template

---

You can read the original blog post by Jacob Hookom (the creator of Facelets) at:

`http://hookom.blogspot.com/2005/05/facelets.html`

In JSF 1.x, you could install Facelets, but it wasn't bundled with JSF itself.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Why XHTML Instead of JSP?

- Facelets avoids some of the complexities and conflicts described in Bergsten's *Improve JSF by Dumping JSP* article that result from using JSF and JSP together
- You can write well-formed XHTML documents for your view
- You can alias JSF tags to standard XHTML elements (similar to the **Tapestry** framework) so that you can use standard XHTML design tools and get a better preview of the page

---

You can read the Hans Bergsten article, "Improving JSF by Dumping JSP", at:

`http://www.onjava.com/pub/a/onjava/2004/06/09/jsf.html`

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring the View File Extension

- The standard extension for JSF pages in JSF 2 and later is XHTML, but you can change that if you wish:

```
1     <context-param>
2         <param-name>
3           javax.faces.DEFAULT_SUFFIX
4         </param-name>
5         <param-value>.jsf</param-value>
6     </context-param>
```


Note that in JSF 1.x, you need to specify .xhtml since .jsp is the default there.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a Facelets Application

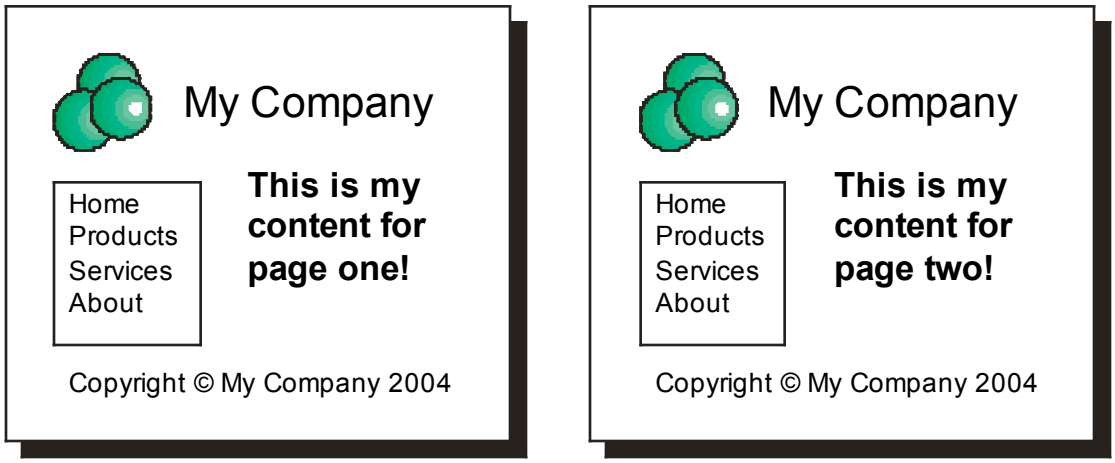
- Steps (in no particular order):
    - Configure the Web application as shown earlier
    - Write one or more XHTML **template** file(s) that define common content and layout
    - Write an XHTML file for each of your pages, specifying a template in each
    - Code JSF managed beans, navigation and so forth in the normal JSF fashion

Using templates is optional, but highly recommended.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Common Layout and Content

- Many websites have many pages that share common content and layout
- Creating this common look manually is tedious and error prone



13 - 6

The pages shown here have exactly the same layout and much of the content is identical. Unless you do something special, maintaining such a site can be tedious. For example, to change the logo on every page or re-position the menu under the logo would be a lot of work if you have many pages.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Writing a Facelets Template

- A **template** lets you define the common layout and content of pages in your Web application
- The template **inserts** the portions of each page that are different from other pages

template.xhtml

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <html xmlns="http://www.w3.org/1999/xhtml"
2      xmlns:h="http://java.sun.com/jsf/html"
3    xmlns:ui="http://java.sun.com/jsf/facelets">
4      <head>
5        <title>
6          <ui:insert name="title">Default title</ui:insert>
7        </title>
8      </head>
9      <body>
10       <h3>
11         <ui:insert name="title">Default title</ui:insert>
12       </h3>
13       <p>
14         <ui:insert name="body">Default body</ui:insert>
15       </p>
16     </body>
17   </html>
```

Writing a Facelets Page

- You base each of your actual pages off of a template – that way, you can effectively update many pages by modifying the template (e.g. change the basic layout)

page1.xhtml

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <html xmlns="http://www.w3.org/1999/xhtml"
2          xmlns:ui="http://java.sun.com/jsf/facelets"
3          xmlns:h="http://java.sun.com/jsf/html">
4    <body>
5
6    This text above will not be displayed.
7
8    <ui:composition template="/template.xhtml">
9
10   This text will not be displayed.
11
12     <ui:define name="title">
13       Page 1 Title
14     </ui:define>
15
16   This text will also not be displayed.
17
18     <ui:define name="body">
19       <h:form id="helloForm">
20         <h:inputText type="text" id="age" value="#{MyBean.age}"/>
21         <br/>
22         <h:commandButton
23            action="success" value="Submit" />
24       </h:form>
25     </ui:define>
26
27   This text will not be displayed.
28
29   </ui:composition>
30
31   This text below will also not be displayed.
32
33   </body>
34   </html>
```

Another Facelets Page

- Here is another page that's based off of the same template:

page2.xhtml

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <html xmlns="http://www.w3.org/1999/xhtml"
2          xmlns:ui="http://java.sun.com/jsf/facelets"
3          xmlns:h="http://java.sun.com/jsf/html">
4    <body>
5
6    This text above will not be displayed.
7
8    <ui:composition template="/template.xhtml">
9
10   This text will not be displayed.
11
12     <ui:define name="title">
13       Page 2 Title
14     </ui:define>
15
16   This text will also not be displayed.
17
18     <ui:define name="body">
19       <h:form id="anotherForm">
20         Your age is:
21         <h:outputText value="#{MyBean.age}"/>
22         <br/>
23         <h:commandButton
24           action="success" value="Back" />
25       </h:form>
26     </ui:define>
27
28   This text will not be displayed.
29
30   </ui:composition>
31
32   This text below will also not be displayed.
33
34   </body>
35   </html>
```

The Facelets Namespace

- The **http://java.sun.com/jsf/facelets** namespace defines several useful elements:
    - **ui:composition** – Defines the content of a templated view
    - **ui:define** – Creates a portion of a templated view (a definition)
    - **ui:insert** – Includes a definition into a template
    - **ui:repeat** – Iterates over a collection
    - **ui:include** – Includes a page fragment (like jsp:include)

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The jsfc Attribute

- Borrowing an idea from the Tapestry framework, Facelets lets you map standard XHTML elements to JSF elements
- This lets you use standard XHTML markup (with a nonstandard attribute) to create views – may work better with design tools and page preview

```
1    <!-- JSF style markup -->
2    <h:commandButton action="success" value="Back"/>
3
4    <!-- XHTML style markup -->
5    <input jsfc="h:commandButton" action="success"
6      value="Back" />
```


Most design tools and page previewers will ignore the nonstandard "jsfc=" attribute and render the page as plain XHTML.

Facelets will convert elements with the "jsfc=" attribute to their corresponding JSF element.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Inlining Text

- In Facelets, you can output values from EL expressions directly, without needing to use the **h:outputText** action
- Whether to use outputText or not is a personal style issue

```
1    <!-- using h:outputText -->
2    <span jsfc="h:outputText" value="#{MyBean.age}"/>
3
4    <!-- inlined text -->
5    <span>#{MyBean.age}</span>
```


Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Facelets and JSTL

- Since Facelets views are not JSPs, they cannot use JSP tag libraries such as the JSTL
- However, Facelets defines a JSTL **namespace** and supports a limited set of JSTL functionality:
    - All of the actions from the **function** library
    - The c:if and c:forEach actions from the **core** library
- In most cases, you should use **ui:repeat** instead of c:forEach

```
1    <html xmlns="http://www.w3.org/1999/xhtml"
2          . . .
3          xmlns:c="http://java.sun.com/jstl/core">
4          xmlns:fn="http://java.sun.com/jstl/functions">
5       . . .
6    </html>
```

13 - 13

There is also limited support for using the c:catch and c:set actions, but the Facelets development team recommends against using them.

It's also possible for the authors of other JSP tag libaries to adapt their code to Facelets. See your library's documentation.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The fundamentals of Facelets
- How to write a simple Facelets program

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to AJAX

- What is AJAX?
- The XMLHttpRequest Object
- Using a Servlet

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is AJAX?

- AJAX (Asychronous JavaScript and XML) uses a combination of existing technologies:
  - JavaScript
  - The XMLHttpRequest Object
  - HTML Document Object Model
- AJAX lets you create Web applications with a higher degree of interactivity



The notion of asynchronous request processing using the XMLHttpRequest object has been around for several years, but the term "AJAX" was coined by Jesse James Garrett of Adaptive Path.

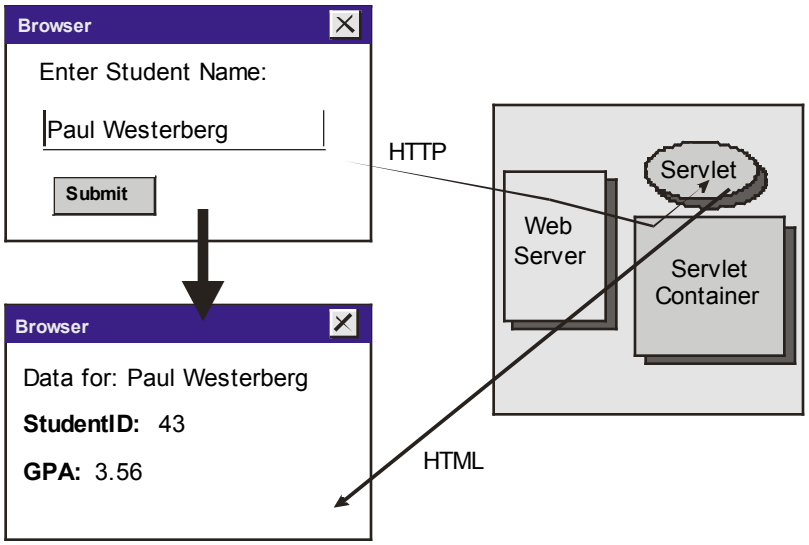
You can read his essay at www.adaptivepath.com/publications/essays/archives/000385.php.

Note that to use AJAX, you're not required to use XML.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Traditional Web Processing

- In the traditional model, users submit pages and Web applications generate complete pages
- This can be inefficient, especially if most of the page is unchanged from one request to the next



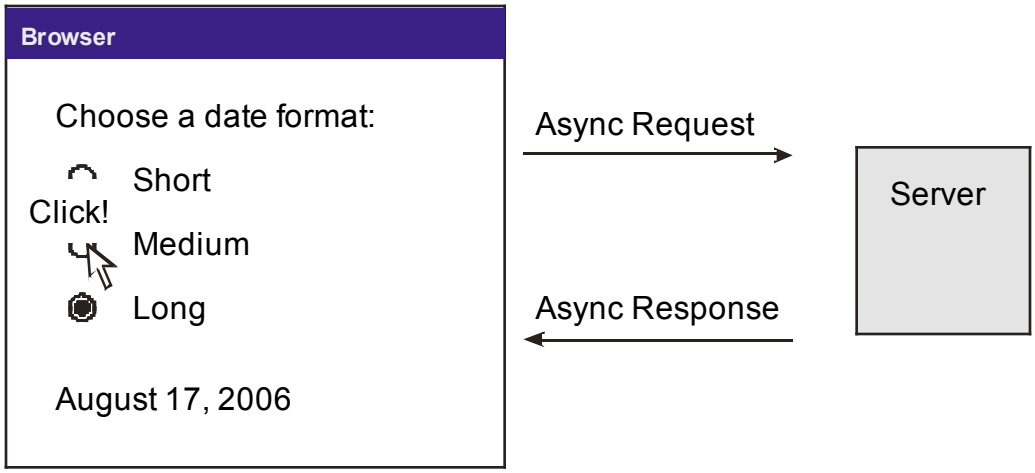
The Web started as a classic two-tier, client-server system in which the Web browser client is nothing more than a display device. All of the processing power and page-generation logic resides on the server.

While this model has been fantastically successful, it's not necessarily the correct fit for all types of applications.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

AJAX Processing

- With AJAX, **any** user action can trigger a request to the server for more or different data
- Since the request is asynchronous, the user can keep working while the request is processed



In an AJAX Web application, JavaScript on the client side can send requests without requiring the user to press a Submit button. In this figure, we show an application that updates a date display whenever the user clicks a radio button. In response, the server sends back a differently formatted string containing the current date.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

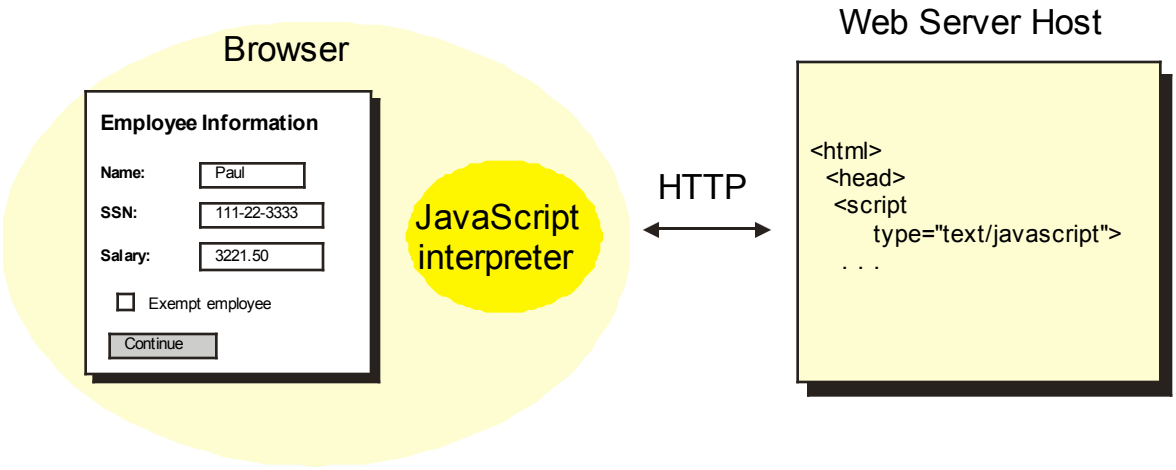
AJAX Applications

- Perhaps the first AJAX application was Microsoft's Outlook Web Access
- Here is a partial listing of others:
    - Flickr
    - Google Maps
    - Google Mail
    - Yahoo Mail

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to JavaScript

- JavaScript is a language, related to Java, that is implemented by most modern Web browsers
- JavaScript programs run on the client, within the browser



You embed JavaScript in your HTML pages, but the scripts actually run in the browser, which is considered "client-side".

JavaScript lets you write Web pages with programmability -- for example, you could write JavaScript to validate the contents of a form when the user presses the submit button. That would save a network round-trip if, for example, the user didn't enter data into a required field.

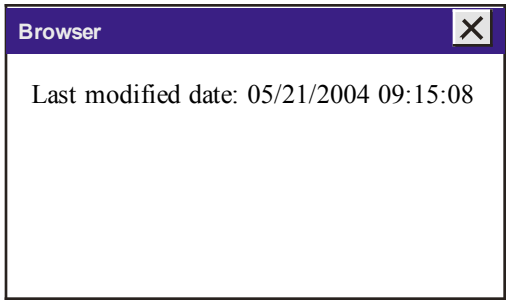
JavaScript is supported by most modern browsers such as Microsoft Internet Explorer and Mozilla Firefox.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining JavaScript

- You can define scripts either within a given page's *head* or *body* elements or by storing the scripts in a separate file

```
<html>
  <body>
    <script type="text/javascript">
      document.write ( "Last modified date: " )
      document.write ( document.lastModified )
    </script>
  </body>
</html>
```



You can embed Javascript either within the "head" or "body" of an HTML page. Here, the script element is within the body since we want the script to generate HTML output.

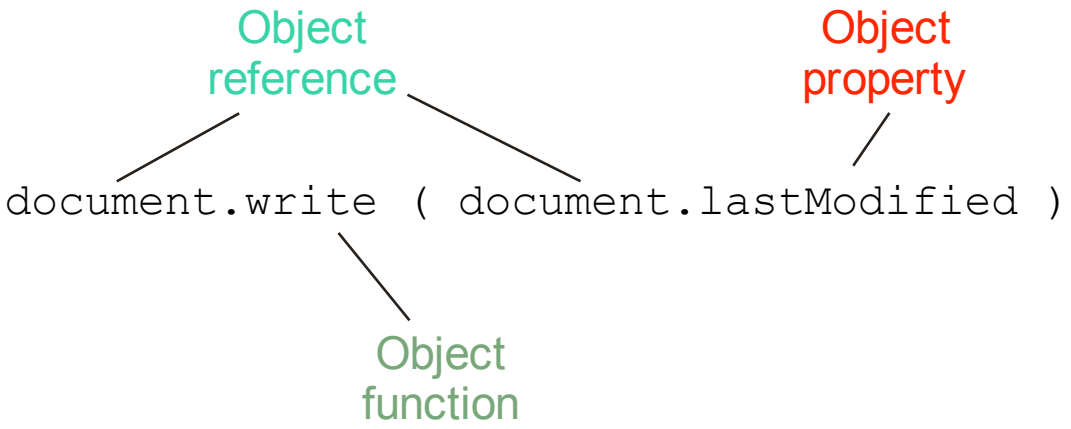
Note that we identify the scripting language using a MIME type -- some browsers support other scripting languages, but Javascript works on most.

You can also store scripts in external files and reference them using the "src" attribute. That lets you reuse the same script in multiple HTML pages.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Object Syntax

- JavaScript supports objects with **properties**, **methods** and **events**
- To access an object's properties, you use the *dot* operator



Here we see a line from a script shown earlier. The "document" reference is the name of the object that represents the entire HTML page.

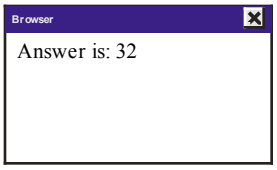
The "write" function lets you insert some text into the document, which the browser eventually displays. You can tell that "write" is a function since it uses parenthesis. The "document.lastModified" is a parameter to the function.

The "document.lastModified" property is a read-only property that tells you when the source HTML page was last changed.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Functions

- To avoid duplicating code in scripts, you can write your own **functions** that you can call from scripts
- You generally define functions in the *head* of the HTML page




```
1    <html>
2      <head>
3        <script type="text/javascript">
4           function sum ( num1, num2 )
5           {
6              return num1 + num2
7           }
8        </script>
9      </head>
10     <body>
11       <script type="text/javascript">
12         document.write ( "Answer is: " )
13         document.write ( sum ( 15, 17 ) )
14       </script>
15     </body>
16   </html>
```


Functions let you organize your scripting code to make it easier to read and maintain.

Functions can accept zero or more arguments. To call a function, you must use parenthesis around the argument values.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Variables

- Any script can define **variables** to hold data
- Unlike in Java, variables do not have declared types
- **Global variables** are defined outside of any function and can be accessed across script elements

```
1    <html>
2      <head>
3        <script type="text/javascript">
4          var mod_date = document.lastModified
5        </script>
6      </head>
7      <body>
8        <script type="text/javascript">
9          document.write ( "Last modified date: " )
10         document.write ( mod_date )
11       </script>
12     </body>
13   </html>
```

Variables are "scratch pad" storage for saving the intermediate results of calculations, user-input data and so forth. Though it's not strictly required, it's a good practice in JavaScript to define variables before use.

By "untyped", we mean that any JavaScript variable can store characters, numbers and so forth. This is different than in Java, where each variable can hold only a single kind of data.

If you define a variable within a function, the variable is considered "local" and can only be used by that function.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Events

- JavaScript defines several **events** that are triggered when the user interacts with page or for page lifecycle events
- You can connect JavaScript functions to an event

Event	Description
onsubmit	The user pressed a form's Submit button
onfocus onblur	The user tabbed to or from an input widget on a form
onclick	The user clicked once on a button, checkbox etc.
onchange	The user changed the text or selection in an text-input field or drop-down list

Event handlers let your JavaScript programs react to user actions.

To handle an event, you write the event's name as an attribute on the source object's definition, for example:

```
<input type="text" onfocus="myfunction()">
```

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JavaScript Events Example

```
1    <html>
2      <head>
3        <script type="text/javascript">
4           function confirmSubmit()
5           {
6              var ok = confirm ( "Really submit?" );
7              return ok;
8           }
9        </script>
10     </head>
11     <body>
12       <form action="http://localhost/test"
13             onsubmit="return confirmSubmit()">
14         <input type="Submit">
15       </form>
16     </body>
17   </html>
```

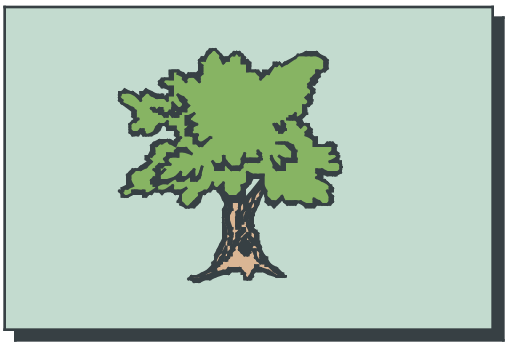
Here we show a JavaScript sample that uses the "onsubmit" event in line 13. When the user submits the form, the browser will call our "confirmSubmit" function, which in turn calls the built-in "confirm" function that displays a message box with OK and Cancel buttons and the specified text. If the user presses OK, "confirm" returns true, otherwise it returns false.

Our "confirmSubmit" function returns the true or false value which the browser then uses to determine whether or not to actually submit the form.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to the HTML DOM

- The **Document Object Model** is a data model and API for accessing the tree-based structure of an HTML or XML document
- The DOM is maintained by the World Wide Web Consortium (W3C)
- You can use JavaScript to invoke the DOM API



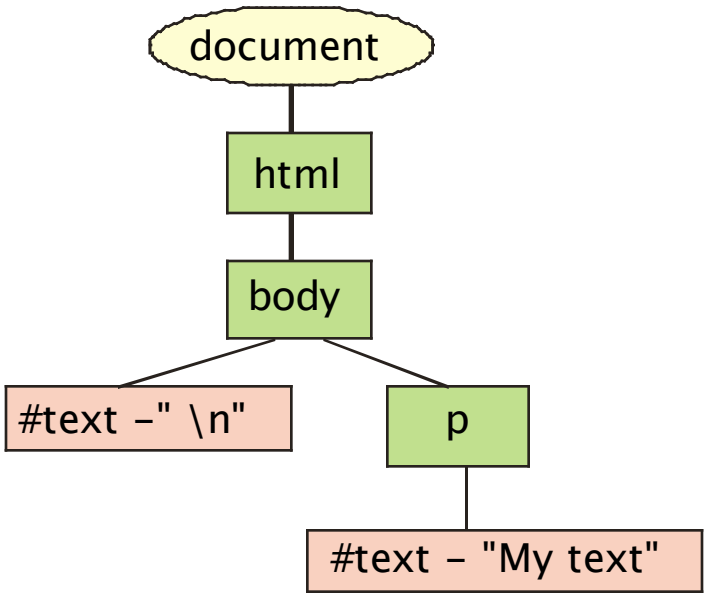
The DOM is a very useful API for working with the components of a Web page or an XML document. AJAX applications can use the DOM to dynamically modify Web pages based on response data from the server.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The DOM Tree

- A well formed HTML document forms a tree, where all *nodes* except the root have a parent
- The DOM provides an API to traverse the tree, either up or down

```
<html>
  <body>
    <p>
      My text
    </p>
  </body>
</html>
```



The "document" object is a pre-defined object that's the root of the DOM tree on every page.

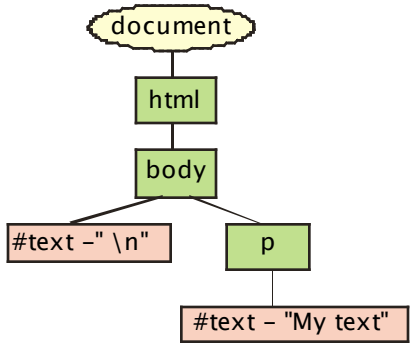
Note that within the body, ALL text is represented in the tree, even text for whitespace characters such as new-lines and tabs.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Accessing Nodes

- The DOM API provides methods for traversing the tree and accessing nodes in the tree

```
1    var htmlEl = document.firstChild;
2    var bodyEl = htmlEl.lastChild;
3    var paraEl = bodyEl.childNodes[1];
4    var textNode = paraEl.firstChild.nodeValue;
5    var textNode1 = paraEl.innerHTML;
6
7    paraEl.firstChild.nodeValue="hello";
```



Here we show some JavaScript that navigates the DOM tree to access nodes. The first line retrieves a reference to the "html" element from the pre-defined "document" object. Note that the "html" element is the first (and only) child of the document.

The second line retrieves the "body" element from the "html" element -- the body is the last (and only) child of the "html" element.

The third line demonstrates using an index to access a particular child node -- note that the first child node (index zero) is a text node that contains only a new-line character.

The fourth and fifth lines do the same thing in this case -- they both retrieve the text from the "p" element. The "innerHTML" property is not actually part of the DOM, but is implemented by most browsers, including later versions of Internet Explorer and Mozilla Firefox. It's nice since it retrieves all the text, even from embedded child elements (e.g if the paragraph had some bold text).

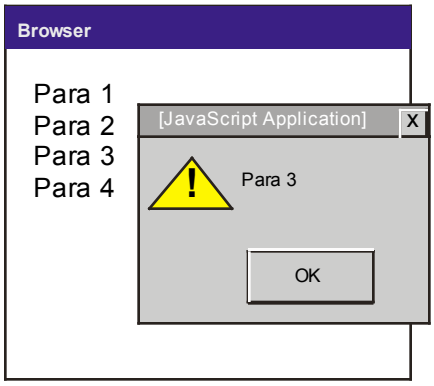
Finally, the last line demonstrates how you can use the DOM to dynamically change the text in a page.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Finding HTML Elements by ID

- HTML lets you assign unique IDs to individual elements

```
1    <html>
2      <head>
3        <script type="text/javascript">
4            window.onload=function()
5            {
6               var p3 = document.getElementById("c");
7               alert(p3.innerHTML);
8            }
9        </script>
10     </head>
11     <body>
12       <p id="a">Para 1</p>
13       <p id="b">Para 2</p>
14       <p id="c">Para 3</p>
15       <p id="d">Para 4</p>
16     </body>
17   </html>
```



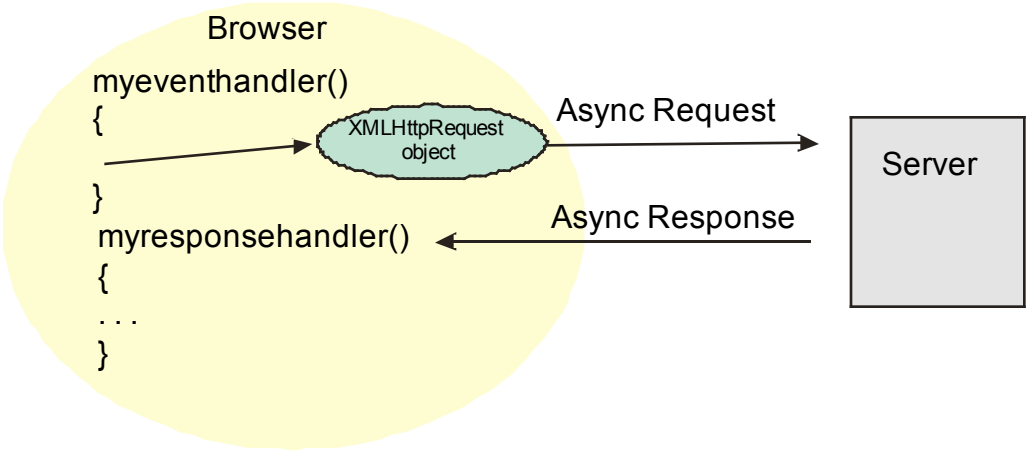
Using the DOM to navigate from parent to child is simple, but can be tedious if you need to access a node that's deep within the tree. To make that easier, the DOM provides a simple search facility that lets you locate elements. This technique depends on the HTML "id" attribute, which lets you assign unique identifiers to elements.

Here we show a JavaScript "onload" event handler that runs when the page loads. In the handler, we use the getElementById method to locate a particular paragraph element. We then display the paragraph's embedded text in a message box.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The XMLHttpRequest Object

- The **XMLHttpRequest** object was first implemented by Microsoft Internet Explorer, but is now a part of other browsers such as Firefox and Opera
- This object defines an API for making remote requests, either synchronously or asynchronously



The XMLHttpRequest object is the foundation of AJAX.

This object lets a JavaScript event handler (e.g. from a radio-button click), send a request to the server and establish another handler to process the response. The response handler can use the DOM to modify the page content based on the returned data. This technique is what lets us write AJAX applications that perform requests without requiring a full page reload.

The XMLHttpRequest object has an interesting history. First implemented by Microsoft Internet Explorer, it has since been "ported" to other browsers and has become a standard part of modern browsers.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Retrieving an XMLHttpRequest Reference

- Retrieving the XMLHttpRequest object reference is somewhat browser dependent

```
1    function getXMLHttpRequest()
2    {
3      var xRequest=null;
4      // handle non-Microsoft browsers
5      if (window.XMLHttpRequest)
6      {
7        xRequest=new XMLHttpRequest();
8      }
9      // handle Microsoft browsers
10     else if (typeof ActiveXObject != "undefined")
11     {
12       xRequest=new ActiveXObject("Microsoft.XMLHTTP");
13     }
14     return xRequest;
15   }
```

14 - 18

While the XMLHttpRequest object itself is mostly standard across browsers, the act of creating the object is different.

Here we show a simple JavaScript method that returns a reference to the object on different browsers.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sending a Request with an Asynchronous Response

- The XMLHttpRequest object supports both HTTP **GET** and **POST** requests
- Here we show using a **GET** with parameters and an asychronous *callback handler*

```
1    var req;
2    function sendAsyncGetRequest(url)
3    {
4      req = getXMLHttpRequest();
5      if (req)
6      {
7        req.open("GET", url, true);
8        req.onreadystatechange = callback;
9        req.send(null);
10     }
11   }
```

```
var url =
  "MyServlet?myparm='abc'";
sendAsyncGetRequest(url);
```

This code listing shows defining "req" as a global JavaScript variable and then a method that sends an asynchronous "get" request to the server.

In line 4, we call the method shown on the last page that creates the XMLHttpRequest object. Assuming that works, we then open a request using "get", the passed URL and specify "true" to configure an asynchronous request. Then in line 8, we register our asynchronous response handler (code shown on the next page). In line 9, we send the request with no additional parameters.

The sample call shown in the figure invokes a server-side program with URL "MyServlet", passing a request parameter named "myparm" with value set to 'abc'.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Sending a Request with an Asynchronous Response, cont'd

```
1    function callback()
2    {
3      if (req.readyState == 4)
4      {
5        if (req.status == 200)
6        {
7          var text = req.responseText;
8          . . . // update the page
9        }
10       else
11       {
12         alert("Problem: " + req.statusText);
13       }
14     }
15   }
```

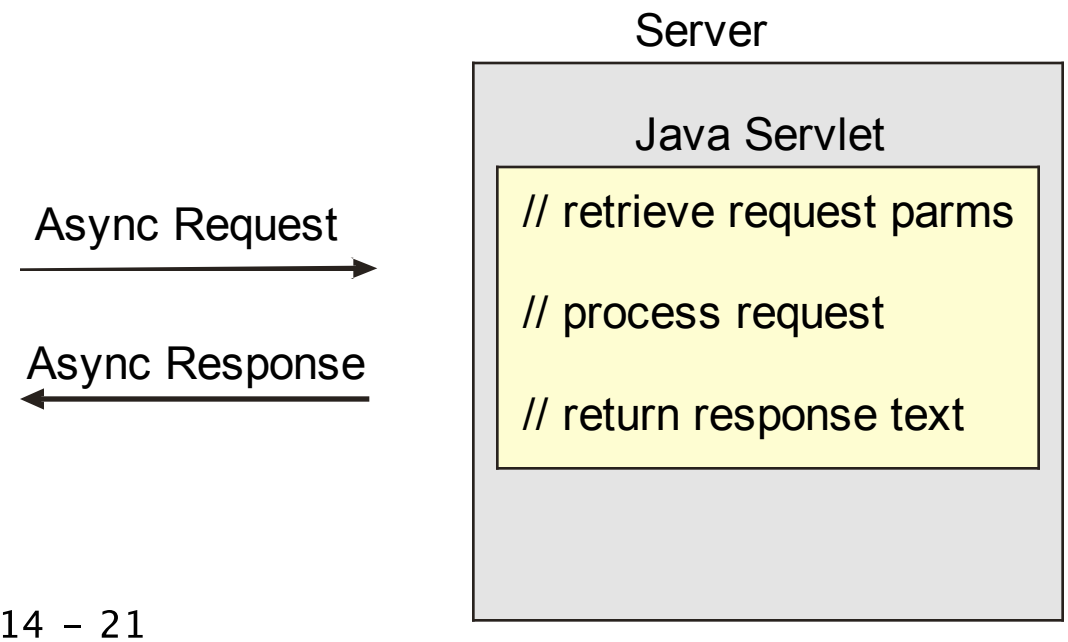

Here we show the JavaScript callback function that we registered to handle the response. It turns out that there are several possible ways that the browser invokes this function for a single request. Here we ignore all of the notifications except for the one that indicates that the request is complete (status = 200). At that point, we could then use the DOM to modify the page to contain the data received from the server, which is referenced by the "text" variable in line 7..

Note that this function uses the global "req" variable we showed on the last page.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a Servlet to Respond

- You can use any sort of server-side Web programming to respond to AJAX requests



AJAX is a client-side technology. On the server, the processing code needs to be able to receive requests, access request parameters and then send back an appropriate response. The main difference between this kind of server processing and traditional Web processing is that the server doesn't send back an entire HTML page in response to an AJAX request.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Updating the Document

- To update the HTML page, the **callback handler** can use the DOM to replace or update a **text node** containing the response text

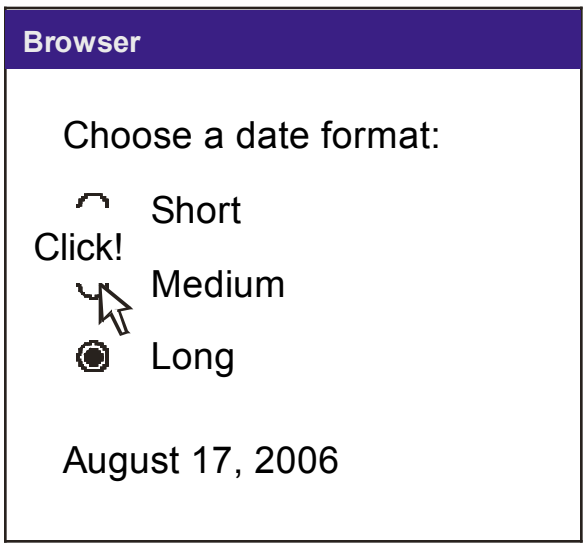
```
1     function callback()
2     {
3       . . .
4       var text = req.responseText;
5       setParagraph(text);
6       . . .
7     }
8
9     function setParagraph(text)
10    {
11      var myPara=document.getElementById('c');
12      myPara.firstChild.nodeValue=text;
13    }
```

After the callback handler retrieves the response, it can use the DOM to change the page to display the returned data. Here we show the callback handler calling a JavaScript function that locates a paragraph by ID and then changes its text-node child to contain the text returned from the server.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example

- The example application lets the user choose a date format and uses a servlet to provide the formatted date



Note that there's no submit button on the page -- instead, the application sends a request whenever the user clicks a radio button.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example Listing

```
 1    <html>
 2      <head>
 3        <title>Current Time and Date</title>
 4        <script type="text/javascript">
 5          var req;
 6
 7          function setDate(text)
 8          {
 9            var dateTxt=
10              document.getElementById('dateTxt');
11            dateTxt.firstChild.nodeValue=text;
12          }
13
14          function changeFmt(text)
15          {
16            var url = "GetDate?fmt=" + escape(text);
17            sendAsyncGetRequest(url);
18          }
14 - 24
```

Here we start the listing for the HTML file that displays the page and contains the JavaScript to perform the AJAX. In line 5, we define a global JavaScript "req" variable to hold the reference to the request. In Lines 7 to 12, we define a function that modifies the paragraph that displays the date.

In lines 14 to 18, the "changeFmt" method sends the request to the "GetDate" servlet. Note that it passes a request parameter named "fmt" and uses the JavaScript "escape" method to ensure that the parameter value complies with the HTTP requirements (i.e. space characters converted to a '+').

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example Listing, cont'd

```
19          function sendAsyncGetRequest(url)
20          {
21            req = getXMLHttpRequest();
22            if (req)
23            {
24               req.open("GET", url, true);
25               req.onreadystatechange = callback;
26               req.send(null);
27            }
28          }
29
30          function callback()
31          {
32            if (req.readyState == 4)
33            {
```

Continuing the HTML listing, the "sendAsychGetRequest" method uses the XMLHttpRequest object to send an asynchronous request and registers the callback function whose listing starts in line 30.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example Listing, cont'd

```
34              if (req.status == 200)
35              {
36                var text = req.responseText;
37                setDate(text);
38              }
39              else
40              {
41                alert("Problem: " + req.statusText);
42              }
43            }
44          }
45
46          function getXMLHttpRequest()
47          {
48            var xRequest=null;
```


Continuing the HTML, lines 34 to 44 complete the callback function. Note that in line 37, after retrieving the response text from the server, the function calls the "setDate" method shown earlier to modify the page.

Line 46 starts the definition of the getXMLHttpRequest function, which retrieves that object in a browser-specific fashion.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example Listing, cont'd

```
49            if (window.XMLHttpRequest)
50            {
51              xRequest=new XMLHttpRequest();
52            }
53            else if (typeof
54              ActiveXObject != "undefined")
55            {
56              xRequest=
57                   new ActiveXObject(
58                     "Microsoft.XMLHTTP");
59            }
60            return xRequest;
61          }
62        </script>
63      </head>
64      <body>
65        <p>Choose format:</p>
```

Continuing the HTML, lines 49 to 61 complete the getXMLHttpRequest function.

Line 64 starts the body of the page.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example Listing, cont'd

```
66        <p>
67          <input type="radio" name="fmt" value="short"
68            onclick="changeFmt('short')">Short
69        </p>
70        <p>
71          <input type="radio" name="fmt" value="medium"
72            onclick="changeFmt('medium')">Medium
73        </p>
74        <p>
75          <input type="radio" name="fmt" value="long"
76            onclick="changeFmt('long')">Long
77        </p>
78        <p id="dateTxt">No date</p>
79      </body>
80    </html>
```


Continuing the HTML, here we define the radio buttons that let the user select a date format. Note that each radio button registers a JavaScript "onClick" handler that invokes the "changeFmt" method shown earlier. Recall that the "changeFmt" method sends an AJAX request to retrieve a formatted date string from the server. Also note that we pass the parameter value, e.g. "short".

Finally, the text that we dynamically modify is the paragraph in line 78. Note that it defines an ID so that the "setDate" method shown earlier can easily find it.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example Listing, cont'd

```
1    package servlets;
2
3    import java.io.*;
4    import java.text.DateFormat;
5    import java.util.Date;
6    import javax.servlet.ServletException;
7    import javax.servlet.http.*;
8
9    public class GetDate extends HttpServlet
10   {
11       protected void doGet(
12               HttpServletRequest request,
13               HttpServletResponse response)
14          throws ServletException, IOException
15       {
16           response.setContentType("text/html");
17           PrintWriter out = response.getWriter();
```

Now we start the listing for the servlet that responds to the AJAX request. This page mostly standard servlet coding that simply sets up to process the request.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example Listing, cont'd

```
18            String fmt = request.getParameter("fmt");
19            DateFormat df = null;
20
21            if (fmt.equals("short"))
22            {
23                df =
24            DateFormat.getDateInstance(DateFormat.SHORT);
25            }
26            else if (fmt.equals("medium"))
27            {
28                df =
29           DateFormat.getDateInstance(DateFormat.MEDIUM);
30            }
31            else if (fmt.equals("long"))
32            {
33                df =
34             DateFormat.getDateInstance(DateFormat.LONG);
35            }
```

Continuing the servlet, in line 18, we retrieve the "fmt" request parameter, which tells the servlet how to format the date string. We then do a bit of Java gymnastics to get a DataFormat object that knows how to format dates.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example Listing, cont'd

```
36              String retStr = "Unknown format";
37              if (df != null)
38                  retStr = df.format(new Date());
39
40              out.print(retStr);
41          }
42      }
```


Finishing the servlet, in line 38, we create a date object containing the current date, and then format it into a string according to request parameter. We can then simply print the formatted date string to the "out" variable, which sends the string back to the AJAX callback handler.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The basics of AJAX
- Fundamental concepts of JavaScript and the HTML DOM

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF and Ajax

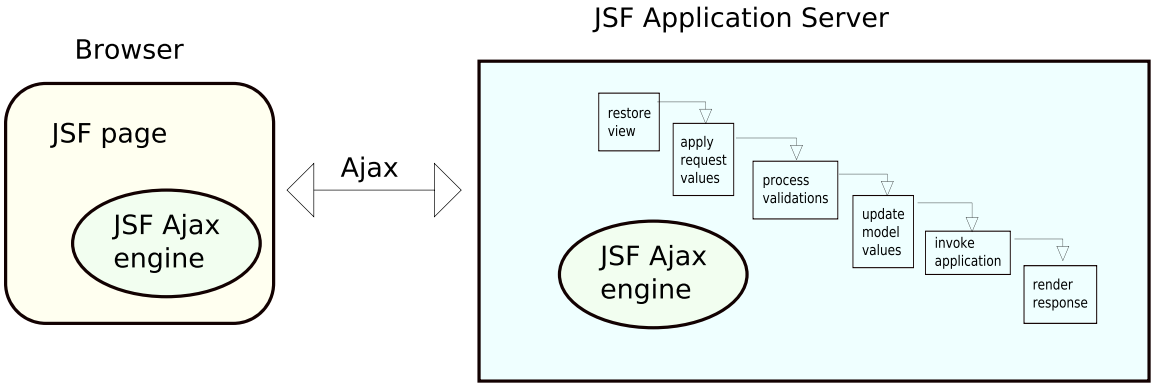
- Partial Page Rendering
- Partial Component Execution

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF and Ajax

- JSF2 includes tags that make it straightforward to add Ajax support to JSF-based applications
- These tags save you from having to write JavaScript to use Ajax



Much of the JSF2 Ajax support is based on the JBoss RichFaces library. Anyone who is familiar with RichFaces will find the JSF2 Ajax easy to learn.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

JSF2 Ajax Support

- The **f:ajax** component tag: Adds Ajax support to existing JSF tags like h:commandButton
- The **jsf.ajax.request** JavaScript API: Provides an API to issue Ajax requests from JavaScript
- The **view** scope: Provides an efficient way to store managed beans for the duration of an Ajax request
- The **PartialViewContext**: Used internally on the server to limit the components that execute during an Ajax request

---

You don't normally need to directly work with the PartialViewContext. Instead, you can specify via f:ajax which components should execute.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The f:ajax Tag

- The **f:ajax** tag lets you configure a parent JSF component to use Ajax
- You must specify the parent's event that will trigger the Ajax request

```
1    <h:selectOneMenu id="make"
2        value="#{car.make}">
3       <f:ajax event="change" />
4       <f:selectItems value="#{car.makes}" />
5    </h:selectOneMenu>
```

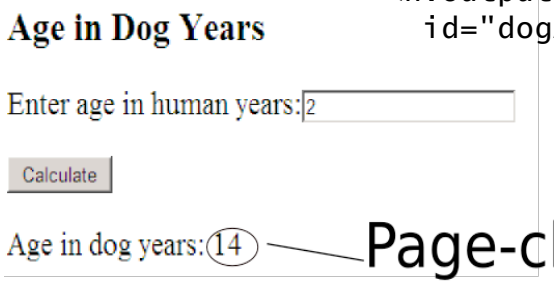
You can use f:ajax to Ajax-ify any JSF component that can raise a JavaScript event. Though the f:ajax tag defines default Ajax events for components, you can explicitly specifiy one as we did here.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Partial Page Rendering

- One feature of Ajax that makes it efficient is **partial page rendering** where you specify which part of a page should change
- JSF provides the **f:ajax** component with which you can specify the **render=** attribute

```
<h:commandButton value="Calculate"
 action="#{ageBean.calc}">
  <f:ajax event="action" render="dogAge"/>
</h:commandButton>
Age in dog years:
<h:outputText value="#{ageBean.dogAge}"
  id="dogAge" />
```



In this example, we have a self-submitting form and the only portion of the page that changes upon form submission is the text for "age in dog years". Note that the application method returns "null" so that we navigate back to the original page.

The render= attribute lets you specify a list of components, separated by spaces.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Partial Page Rendering

- Controlling which parts of a page render during Ajax requests is an important concept for efficiency and performance:
    - Reduce server-side processing
    - Reduce network bandwidth
- During an Ajax request, on the server, JSF suppresses rendering of components NOT specified in a re-render list

---

The client-side JavaScript only updates the DOM for elements specified in the render list.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Partial Page Rendering Options

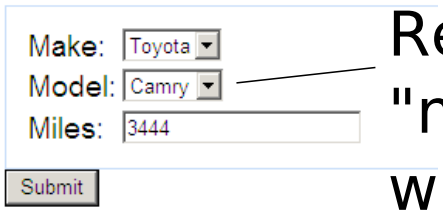
- The **render=** attribute supports these values:

Value	Description
@all	All components render
@none (default)	No components render
@this	Only current component renders
@form	All components in the form render
id1 id2 ...	Space-separated component ID list
#{...}	A JSF expression that evaluates to a render list

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Ajax Requests Without Form Submission

- The example on the last page did a partial-page update, but required the user to press a "submit" button
- Ajax applications often send requests based on a user interaction (e.g. typing a key, choosing an item from a list) that don't involve a page submission



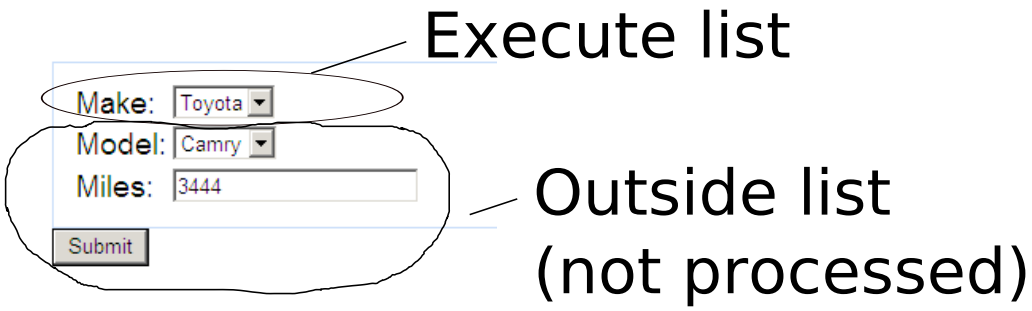
Repopulate "model" list when user chooses "make"

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Introduction to Execute Lists

- When a Ajax request is submitted, you can specify an **execute list** of components on the page that should run through the entire JSF lifecycle

```
1     <h:selectOneMenu id="make"  value="#{car.make}">
2       <f:ajax event="change" execute="make"
3            render="model" />
4       <f:selectItems value="#{car.makes}" />
5     </h:selectOneMenu>
```



Re-rendering and execute lists are related, but conceptually different. Rendering is all about how page components generate output and how the client-side DOM is updated. Execute lists lets you control which components on the server actually run during certain JSF phases during the request.

When we say processing, that includes the JSF phases: Restore View, Apply Request Values, Process Validations (which includes conversion), and Update Model Values.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Execute List Options

- The **execute=** attribute supports these values:

Value	Description
@all	All components execute
@none	No components execute
@this (default)	Only current component executes
@form	All components in the form execute
id1 id2 ...	Space-separated component ID list
#{...}	A JSF expression that evaluates to an execute list

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Rendering vs Executing

- These concepts are related: we use both to optimize Ajax processing
- Rendering controls which server components actually generate output into the response and which parts of the page the client-side DOM updates
- Execute lists control which server components run through the Request Values, Process Validation and Update Model phases

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Using a JSF Phase Listener

- A **phase listener** is a JSF listener class that lets you hook into the JSF request lifecycle
- Using a phase listener is helpful when you are learning/debugging Ajax4jsf
- To write a phase listener:
    - Write a class that implements the **PhaseListener** interface
    - Register the class in faces-config.xml

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- The basics of Ajax with JSF
- About partial page rendering and execute lists

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The OWASP Top Ten

- The Top Ten Vulnerabilities for 2010
- Mitigating the Vulnerabilites

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is OWASP?

- From http://www.owasp.org:
- The Open Web Application Security Project (OWASP) is a 501c3 not-for-profit worldwide charitable organization focused on improving the security of application software. Our mission is to make application security visible, so that people and organizations can make informed decisions about true application security risks. Everyone is free to participate in OWASP and all of our materials are available under a free and open software license



You can read more about the OWASP organization at:

`http://www.owasp.org/index.php/OWASP:About`

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The OWASP Top Ten for 2010

- OWASP publishes a list of the top-ten current application vulnerabilities, most of which are geared toward Web applications:

- A1 – Injection
- A2 – Cross-Site Scripting (XSS)
- A3 – Broken Authentication and Session Management
- A4 – Insecure Direct Object References
- A5 – Cross-Site Request Forgery
- A6 – Security Misconfiguration
- A7 – Insecure Cryptographic Storage
- A8 – Failure to Restrict URL Access
- A9 – Insufficient Transport Layer Protection
- A10 – Unvalidated Redirects and Forwards

The Top Ten for 2010 supercedes the previous list, which was released in 2007.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Injection

- Injection flaws occur when untrusted data is sent to an interpreter (e.g. SQL database) as part of a command or query
- An attacker can trick the interpreter into executing unintended commands for accessing unauthorized data

```
HTML Form                Program
---------                -------
                       String s =
                          req.getParameter("serial");

<input type="text"     String SQL =
  name="serial">         "SELECT FROM Users WHERE
                         serial='" + s + "'";

                       ResultSet rs =
                          stmt.executeQuery(SQL);
```

The most common form of injection is SQL injection, but the problem's not limited to databases. For example, LDAP directories could be at risk.

For a good explanation of how SQL injection works, see:

`http://unixwiz.net/techtips/sql-injection.html`

In the code shown here, an attacker could "inject" unintended SQL by entering a string that contained SQL "escape" syntax.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mitigating Injection in Java

- Your primary protection against SQL injection is to use JDBC *PreparedStatements* to construct SQL from user input

```
HTML Form                 Program
---------                 -------
                        String s =
                           req.getParameter("serial");

<input type="text"      String SQL =
  name="serial">          "SELECT FROM Users WHERE
                          serial=?";

                        PreparedStatement stmt =
                           conn.prepareStatement(SQL);
                        stmt.setString(1, serial);
                        ResultSet rs =
                           stmt.executeQuery();
```

This technique is effective since the JDBC driver validates SQL before executing it, thus reducing the likelihood of an injection.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Cross-Site Scripting

- XSS flaws occur when a Web application passes untrusted data to a Web browser
- Attackers use XSS to execute scripts in the user's browser to hijack session, deface Web sites or redirect the user to malicious sites

```
JSP With Input Form
-------------------
<form action="output.jsp">
  Enter your name: <input type="text" name="name">
  <input type="submit" value="Submit">
</form>

Output JSP
----------

You entered: ${param.name}
```

User Input:

<script type="text/javascript">alert('Got you!');</script>

Note that the user input contains a simple JavaScript. This script is harmless (it displays an alert box), but attackers can write much more malicious scripts to hijack sessions and so forth.

The problem here is that the JSP expression (${...}) simply displays the input text within a generated output page.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mitigating XSS in Java Web Apps

- If your Web application accepts input from the user and then redisplays it in a Web page, you should ensure that *XML escaping* is enabled
- Note that the JSP *expression language* does NOT escape XML by default

```
Output JSP
----------

Bad:  ${param.name}
Good: ${fn:escapeXml(param.name)}
Good: <c:out value="${param.name}" />
```


A JSP can minimize the likelihood of XSS by "escaping" untrusted input before outputting it. Note that the "fn:escapeXml" and "c:out" are functions from the JSP Standard Tag Library (JSTL) and that "c:out" escapes content by default. Be aware, however, that this is not complete protection, since an attacker could submit the input encoded in hexadecimal. So before escaping, you should first decode any hexadecimal text in the user input. See the following article for details:

http://today.java.net/article/2005/09/19/handling-java-web-application-input-part-2

Also note that if you are using JavaServer Faces (JSF) with the Facelets view technology, it automatically escapes output, too.

You can also encounter XSS if your application has a servlet that directly outputs untrusted user input.

See:

http://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet a

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

for more details.

Broken Authentication and Sessions

- This vulnerability occurs when applications do not correctly implement authentication and session management
- An attacker can compromise passwords or session tokens

```
http://www.goliath.com/myServlet;jsessionid=863F373633D316
```


HTTP is a "stateless" protocol, so Web applications that need to maintain state (e.g. a shopping cart) typically use browser cookies to maintain state between requests. But some developers use also "URL rewriting" in case the user disables browser cookies; if so, then the session information appears in the URL itself.

This potentially opens the application to attack since an attacker can see the session ID via a network sniffer or XSS. An attacker can use the sessionID to mount an attack. For more details in the Java world, see:

```
http://www.javapractices.com/topic/TopicAction.do?Id=226
```

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mitigating Broken Sessions in Java Web Apps

- One possible vulnerability is Web applications that use *URL rewriting* to maintain session state
- Other vulnerabilities relate to not properly invalidating sessions at the end of a flow or using long session timeouts
- Another attack involves applications that store user passwords in the clear on a server-side database - once an attacker compromises the database, they have access to all users' passwords

---

See:

`http://www.owasp.org/index.php/ASVS`

for more details.

Java Web developers should avoid using URL rewriting and instead use browser cookies to maintain state.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Insecure Direct Object References

- This vulnerability occurs when a program exposes a reference to an internal "object" such as a file, directory or database key
- Without proper access protection, an attacker can use the "object" to access unauthorized data

```
// URL from authorized access

http://www.goliath.com/myServlet;accountID=1234

// Attack to access unauthorized account

http://www.goliath.com/myServlet;accountID=7777
```


The idea here is that an attacker discovers a reference and then attempts access using forged reference, attempting to access unauthorized information.

In the example shown here, after the attacker learns that '1234' is a legal account, the attacker attempts to access acount number 7777.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mitigating Insecure Object Vulnerabilities

- Applications should not use explicit database keys in user interfaces, but should use "magic numbers" that the application maps to the actual keys
- Developers and administrators should ensure that any file or directory access from an application refers to a properly secured file system

---

See

`http://www.owasp.org/index.php/ESAPI`

for more details.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Cross-Site Request Forgery

- A CRSF attack forces a logged-in victim's browser to use a forged request
- The attacker can supply parameters on the forged request to cause the vulnerable application to perform unintended actions

This attack often occurs on Internet "forum" sites where users can enter "img" tags but not script, or via spam email.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

CRSF Example

Step 1: Sue logs into bank Web application, which establishes a session

Step 2: Sue browses to a Web page containing a form to transfer money to an account

Step 3: Sue submits the form

```
POST  http://mybank.com/transfer
. . .
acct=1234
amt=100
```

Web app uses Sue's session to authenticate and performs transfer

Step 4: Evil Sam sends Sue an HTML email with the following content:

```
<img width=1" height="1" src="http://mybank.com/transfer?acct=7777&amt=1000">
```

Browser in HTML email client sends a GET request using Sue's session
Insecure Web application transfers $1000 to Evil Sam's account

---

The email contains a virtually invisible picture that will not render in the browser, but the browser WILL make the request specified by the "src" attribute, triggering the bank transfer.

Note for this attack to work, Evil Sam must know a valid account number and must trick Sue into opening the email before she logs out of the bank application or the session times out.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mitigating CRSF

- Don't process GET requests when POST is expected (but recognize that attackers can forge POSTs, too)
- To effectively mitigate CRSF, applications must ensure that all forms contain a unique security token that that the application validates when the request arrives
- Some Java application servers, e.g. Tomcat version 7 and IBM WebSphere (via the *ProjectZero* initiative) provide automatic, configurable CRSF protection

---

See

```
http://en.wikipedia.org/wiki/Cross-site_request_forgery
http://www.owasp.org/index.php/ESAPI
http://www.projectzero.org/
http://www.developer.com/java/ent/article.php/3891171/Tomcat-7-Debuts-for-Java.htm
```

for more details.

The OWASP group provides a CRSF testing tool at

```
http://www.owasp.org/index.php/CSRFTester
```

to help detect CRSF vulnerabilities.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mitigating CRSF, cont'd

Step 1: Sue logs into bank Web application, which establishes a session

Step 2: Sue browses to a Web page containing a form to transfer money to an account

Application generates a unique "token" and stores it in Sue's session on the server and as a hidden field in the form sent back to Sue's browser

Step 3: Sue submits the form

```
POST  http://mybank.com/transfer
. . .
acct=1234
amt=100
token=7hjfsadf834343
```

Web app first validates token against the value stored in Sue's session and only then performs transfer

Step 4: Evil Sam sends Sue an HTML email with the following content:

```
<img width=1" height="1" src="http://mybank.com/transfer?acct=7777&amt=1000">
```

Browser in HTML email client sends a GET request using Sue's session
Web application rejects request since it doesn't contain the required token

---

By requiring a unique security token associated with the form and the legitimate user's session, we make it difficult for Evil Sam to forge a request (he would also need to guess the security token).

To further reduce risk, the application can "age" the token to minimize the time window in which the application is vulnerable to CRSF.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Security Misconfiguration

- This vulnerability occurs when applications use frameworks, libraries, application servers and databases that contain vulnerabilities

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mitigating Security Misconfiguration

- Ensure that all accessory software is patched and up to date and that there's a well defined process for updating
- Ensure that unnecessary services, ports and unused pages are disabled
- Ensure that security settings in your application server are configured properly

---

See:

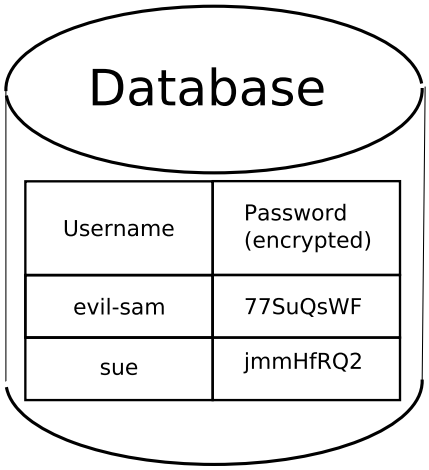
`http://www.owasp.org/index.php/Configuration`

for more details.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Insecure Cryptographic Storage

- This vulnerability occurs when applications do not properly protect sensitive data, such as credit card numbers and passwords



Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mitigating Insecure Storage

- Ensure that sensitive data is encrypted using strong encryption algorithms
- Ensure that access to sensitive data is limited to authorized users

---

See

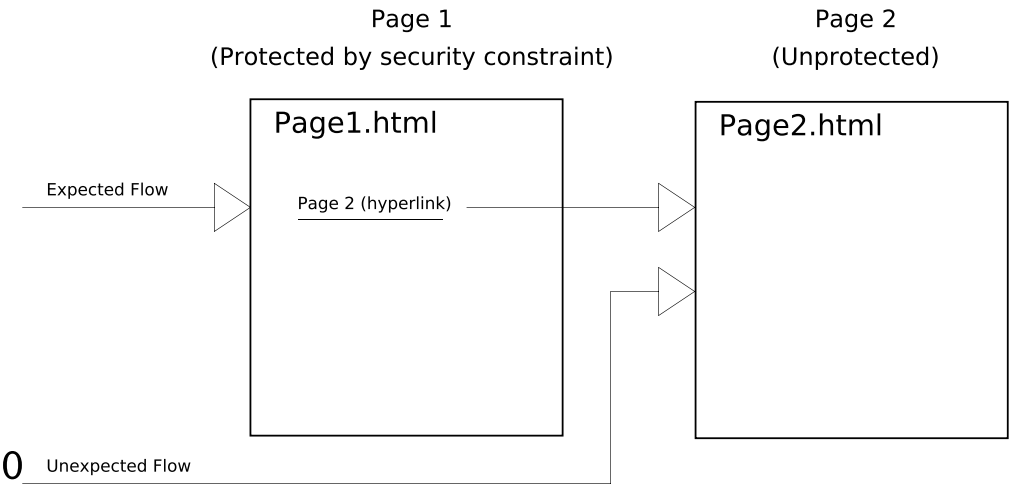
`http://www.owasp.org/index.php/ASVS`

for more details.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Failure to Restrict URL Access

- This vulnerability occurs when applications fail to perform access control checks whenever protected resources are accessed, but only protect links and submit buttons
- An attacker can forge URLs that access restricted pages and bypass access control checks



Many Web server infrastructures, including Java Enterprise Edition (JEE), let you configure security constraints to restrict access to pages in a Web application.

This vulnerability results from the failure to protect all sensitive pages, but to instead only protect the "landing" pages of the application.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mitigating Failure to Restrict URL Access

- Deny all access to protected pages by default and use explicit grants for specific users on specific pages
- Use role-based security to make restriction easier to maintain when user population is dynamic

---

See

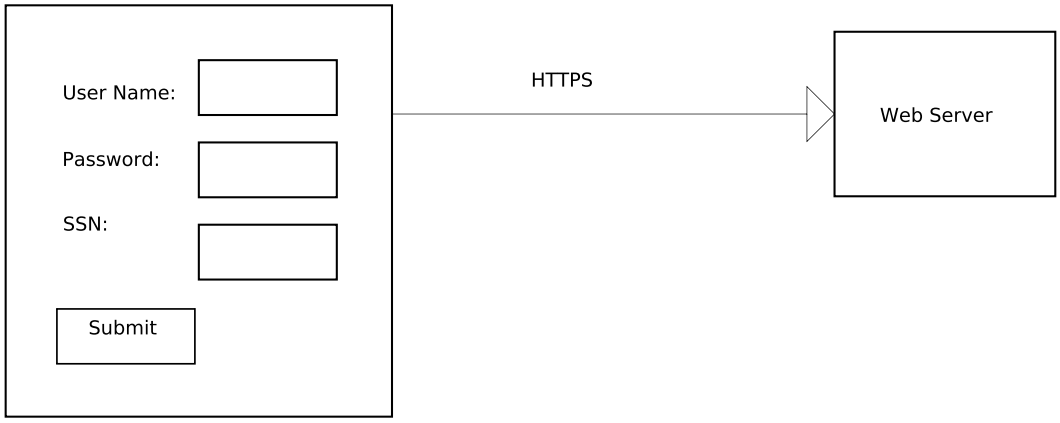
`http://www.owasp.org/index.php/Guide_to_Authorization`

for more details.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Insufficient Transport Layer Protection

- This vulnerability occurs when applications fail to use strong encryption (e.g. SSL) as sensitive data passes over an insecure network



Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mitigating Insufficient Transport Layer Protection

- Use SSL whenever sensitive data is transmitted over an insecure network, especially for login credentials
- Avoid using mixed secure and insecure content on a given page
- Ensure that the SSL certificates are up to date, use strong encryption and passwords and are properly configured

---

See

`http://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet`

for details.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Unvalidated Redirects and Forwards

- This vulnerability occurs when a Web application redirects or forwards to another page using untrusted data to determine the destination
- An attacker can redirect to phishing or malware sites or forward to unauthorized pages

```
Java Servlet Code
-----------------

String dest = req.getParameter("form_field");

. . .

resp.sendRedirect(dest);
```


Forwarding and redirecting are Web application techniques to transfer control programmatically. Forwarding is generally limited to within a Web application, while redirection is part of the HTTP specification and can transfer to any URL.

In the example shown here, an attacker could provide a URL in a form parameter that would cause the servlet to redirect the user to a potentially malicious site.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Mitigating Unvalidated Redirects and Forwards

- Avoid using forwarding and redirecting unless absolutely necessary
- If an application uses input from the user to determine the destination, validate the destination address

---

See

`http://www.owasp.org/index.php/Open_redirect`

for more details.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The OWASP Java Project

- The OWASP group has a *Java Project* whose goal is to enable Java and J2EE developers to build secure applications efficiently
- This project has the following subcategories:
    - J2EE Security for Architects
    - J2EE Security for Developers
    - J2EE Security for Deployers
    - J2EE Security for Security Analysts and Testers

This OWASP project has specific details for security in the JEE and JSE environments. See

`http://www.owasp.org/index.php/Category:OWASP_Java_Project`

for more information.

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Chapter Summary

In this chapter, you learned:

- About The Open Web Application Security Project
- The OWASP Top Ten Vulnerabilities
- How to mitigate the vulnerabilities

Copyright © Descriptor Systems, 2001-2011. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum