

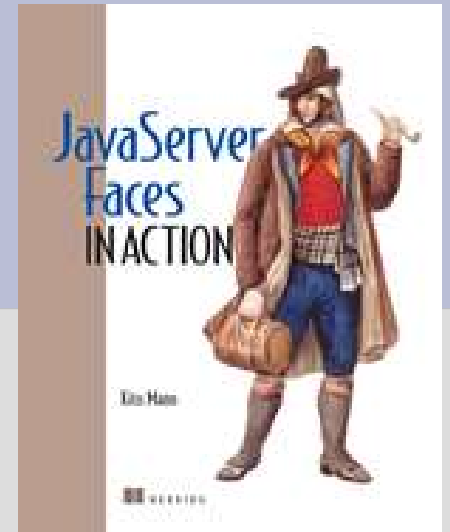
All about JavaServer Faces

Component-oriented, rapid web development

C++ and Java SIG
New York PC User's Group
April 1st, 2004

Kito D. Mann
kman@virtua.com
<http://www.manning.com/catalog/view.php?book=mann>
<http://www.jsfcentral.com>

Kito D. Mann



- Hands-on enterprise architecture consultant and author
- Author, JavaServer Faces in Action (Manning Publications, May/June, 2004)
- Founder, JSF Central – development community for JavaServer Faces with news, FAQ, links, and so on (www.jsfcentral.com)
- Experience with Java since its release in 1995

Agenda

- Introduction
- Key concepts
- Developing a JSF application
- JSF inside of an IDE
- Developing custom components
- Extending JSF
- Wrap up and book giveaway

Agenda

- ***Introduction***
- Key concepts
- Developing a JSF application
- JSF inside of an IDE
- Developing custom components
- Extending JSF
- Wrap up and book giveaway

It's a RAD-ical world

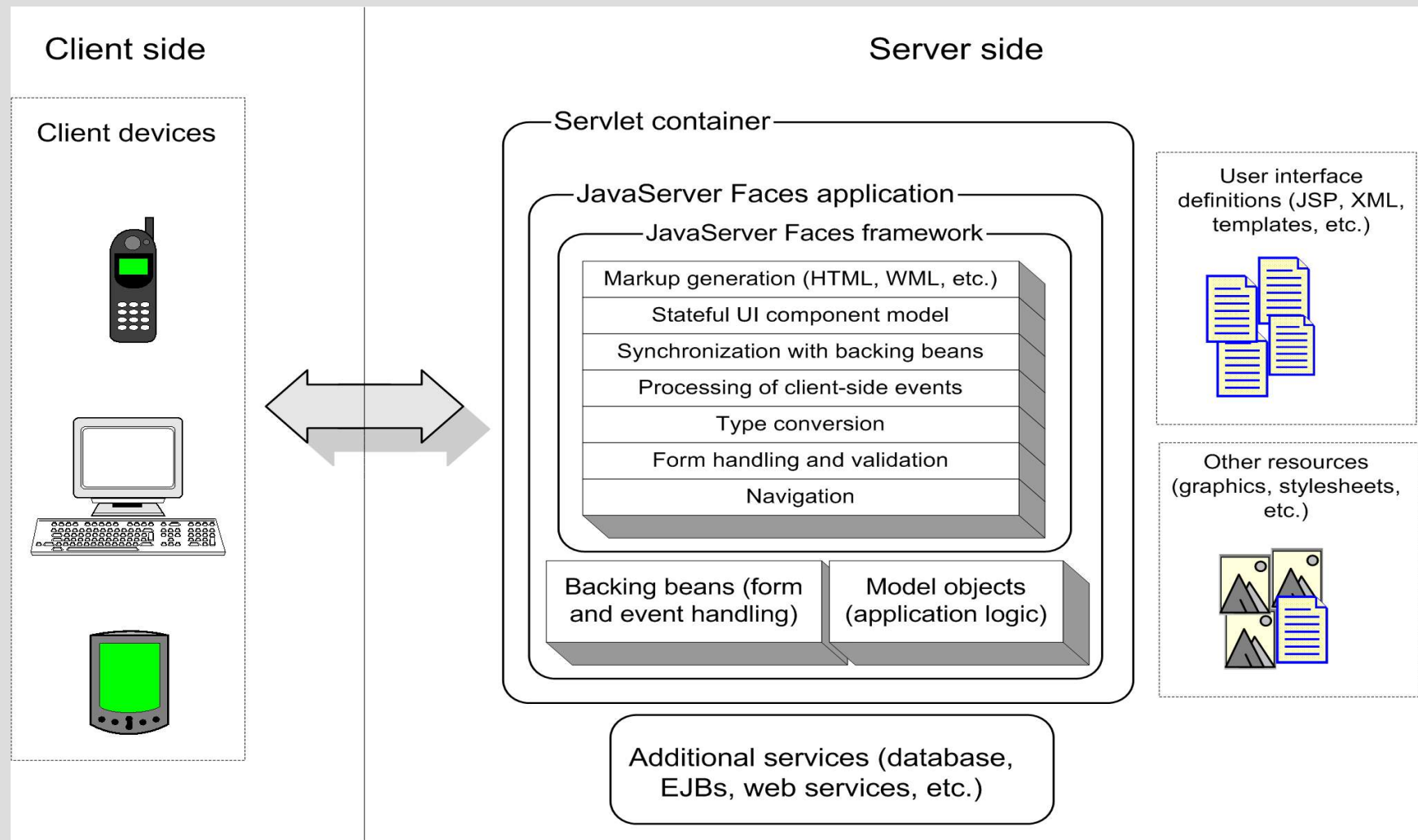
- RAD = Rapid Application Development
- Early 90s, height of client/server
- Easy-to-use, visual, component-oriented tools
 - PowerBuilder, Visual Basic, Delphi
- Defined user interface (UI) component and event model, standard UI components, and application infrastructure
- Closest thing in Java world is Swing (and SWT) for desktop
- No standard solution in Java web world

What is JavaServer Faces?

- Standard web user interface framework for Java
- Defines UI component and event model, standard UI components, and application infrastructure
- UI components live on the server
- Client-generated events are handled on the server
- Can automatically synchronize UI components with application objects
- Extensive tool support (Sun, IBM, Oracle, others)
- Enables RAD-style approach to Java web development
- Sets stage for third-party UI component market
- Works with JSP, but does not require it

What is JavaServer Faces?

JSF Application Architecture

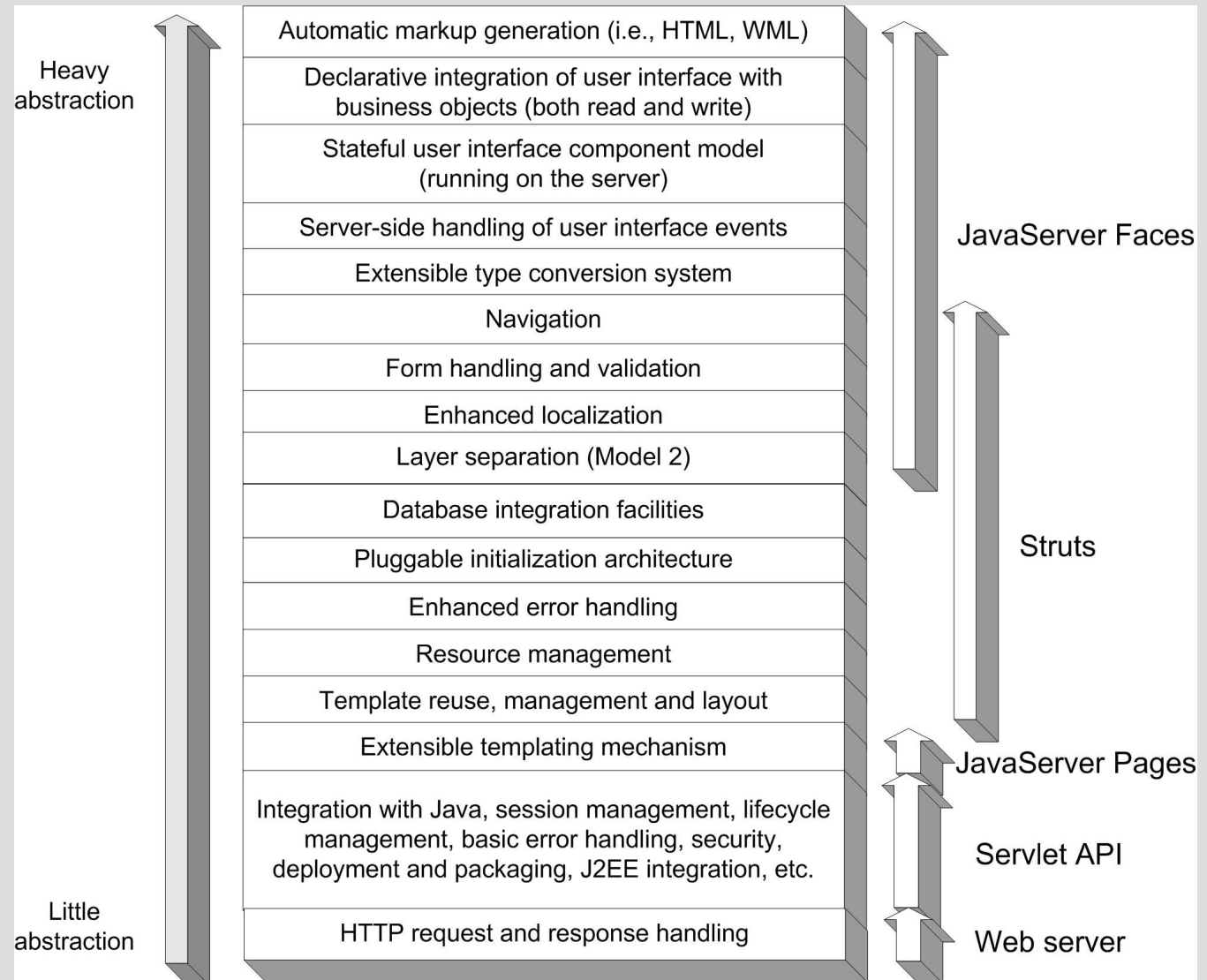


Why do we need another Java web framework?

- Two types of frameworks:
 - Foundation
 - Struts, WebWork, Maverick, many others
 - User interface
 - Tapestry, SOFIA, many others
- Over 30 different Java web development frameworks available
- Framework paralysis!
- JSF is a **standard** best-of-breed framework
- Java needed a competitor to Microsoft's ASP.NET WebForms

JSF and Struts

- JSF is UI framework
- Services overlap with Struts
- Can be used with Struts

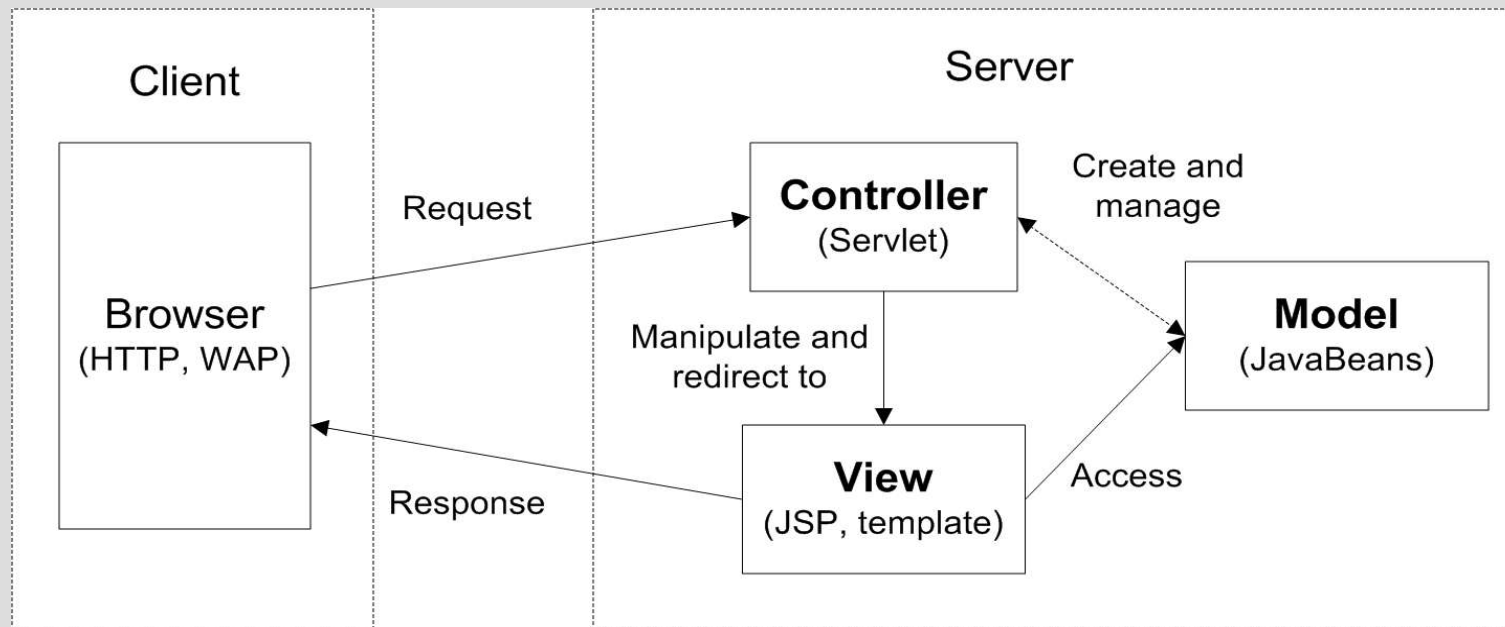


Web frameworks use the MVC pattern

- Model-View-Controller (MVC)
 - Model is application and data
 - View is user interface
 - Controller responds to user input and talks to model
- Classic MVC assumes fine-grained events
- Not suitable for web (UI in a browser; too many round trips)

Web frameworks use the MVC pattern

- Web variation is “Model-2”
 - View is JSP or other display technology (Velocity, XSLT, and so on)
 - Controller is a servlet
 - Model are JavaBeans



JSF is closer to classic MVC

- Events are more fine-grained (but not as fine-grained as desktop applications)
- Controller code can handle UI events directly
- Controller code has direct access to UI components (on server)

Agenda

- Introduction
- ***Key concepts***
- Developing a JSF application
- JSF inside of an IDE
- Developing custom components
- Extending JSF
- Wrap up and book giveaway

Key JSF concepts

- User interface components
- Renderers
- Backing beans
- Validators
- Converters
- Events and listeners
- Expression language
- Messages
- Navigation

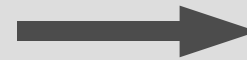
User interface components

- Objects that manage interaction with a user
- May be responsible for its own display, or may delegate display to a renderer
- Stored in a tree on the server (“view”)
- Retain state inbetween client requests
- Standard components: text box, panel, label, data grid, graphic, listbox, radio button, check box, and so on
- Other possibilities: toolbar, menu, RSS viewer, tabbed pane, file upload, and so on

User interface components

- UI components can be declared in markup:

```
<h:inputText id="helloInput" value="default"  
            required="true">
```



default

- and manipulated on server in Java code:

```
...  
HtmlInputText input = (HtmlInputText)event.getSender();  
input.setDisabled(true);  
input.setStyle("color: blue");  
...
```

Renderers

- Responsible for encoding and decoding components
- *Encoding* displays the component
- *Decoding* translates the user's input into component values or events
- Grouped into render kits
 - JSF ships with an HTML 4.01 render kit
 - Render kits can implement a look and feel (“skin”)
 - Render kits can target a specific device (phone, PC) or markup language (WML, HTML, SVG)
 - The render kit can be changed on the fly

Backing beans

- Collect form input from components
- Properties can be synchronized with component values
- Can reference and manipulate UI component instances
- Handle UI events
- A combination of Struts ActionForms and Struts Actions
- Conceptually similar to code-behind classes in ASP.NET WebForms
- Usually talk to model objects to execute actual business logic

Backing beans

- You can bind a component's value to a backing bean property:

```
<h:outputText id="helloBeanOutput"  
              value="#{helloBean.numControls}"/>
```

- You can also bind a component instance to a backing bean property:

```
<h:panelGrid id="controlPanel" binding="#{helloBean.controlPanel}"  
            columns="20" border="1" cellspacing="0"/>
```

Backing beans

```
public class HelloBean {
    private int numControls;
    private HtmlPanelGrid controlPanel;

    public HtmlPanelGrid getControlPanel() {
        return controlPanel;
    }
    public void setControlPanel(HtmlPanelGrid controlPanel) {
        this.controlPanel = controlPanel;
    }
    public int getNumControls() {
        return numControls;
    }
    public void setNumControls(int numControls) {
        this.numControls = numControls;
    }
    ... // event listener and/or validation methods
}
```

Backing beans

- Configured using *managed bean creation facility*
- Facility can be used for model objects as well
- Configured in JSF configuration file
- Object will be created automatically if it doesn't exist
- Example:

```
<managed-bean>
  <description>The one and only HelloBean.</description>
  <managed-bean-name>helloBean</managed-bean-name>
  <managed-bean-class>org.jia.hello.HelloBean
</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Validators

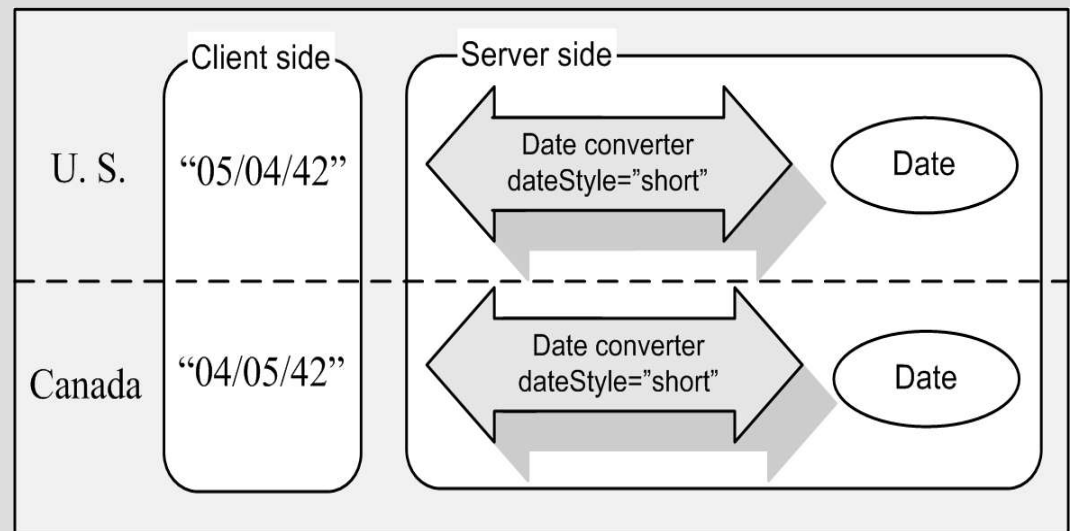
- Validators verify that a component's value is acceptable
- A UI component can be associated with one or more validators
- Validation can also be handled by backing bean methods
- JSF includes standard validators for checking range and length
- Example:

```
<h:inputText id="helloInput" value="# {helloBean.numControls}"  
              required="true">  
    <f:validateLongRange minimum="1" maximum="500"/>  
</h:inputText>
```

Converters

- Convert value of the component to and from a String for display
- Perform formatting or localization
- Standard converters for all basic Java data types
- Example:

```
<h:outputText value="# {user.dateOfBirth}">  
  <f:convert_datetime type="date" dateStyle="short"/>  
</h:outputText>
```



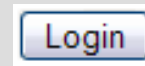
Events and listeners

- Uses JavaBean event model (like Swing)
- Objects create events which are consumed by listeners
- Listeners can be implemented as backing bean methods (unlike Swing) or separate listener classes (like Swing)
- Action methods – special listeners that perform logic and impact navigation
- Standard events
 - Action events (user clicked on a button or link)
 - Value-change events (value of control changed)
 - Data model events (new row in data set selected)
 - Phase events (used when processing a request)

Events and listeners

- Example: When a user clicks on the button, an action event is fired, and the action method is executed.

```
<h:commandButton type="submit" value="Login"  
                action="#{loginForm.login}"/>
```



```
public class LoginForm {  
    public String login() {  
        if (...) // login is successful  
        { return "right on"; }  
        else  
        { return "no way"; }  
    }  
    ...  
}
```

The JSF expression language

- Used to associate UI components with backing beans and model objects
- Based on EL included in JSP 2.0
- Properties are referenced with *value binding expressions*: `{myBean.myProperty}`
- Methods are referenced with *method binding expressions*: `{myBean.myMethod}`
- Supports mixed literal values and implicit variables
- Can interact with same objects as JSP 2.0 and JSTL tags (or rest of web application)

Messages

- Built-in support for application messages
- Messages created by validators, converters, or application code
- Can be displayed by UI components

Navigation

- Full support for declarative navigation
- Outcome of action methods used to select next page
- Eliminates need for Java code or JSPs to know file names

```
<navigation-rule>
  <from-view-id>/login.jsp</from-view-id>
  <navigation-case>
    <from-outcome>right on</from-outcome>
    <to-view-id>/mainmenu.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>no way</from-outcome>
    <to-view-id>/login.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>mainmenu</from-outcome>
    <to-view-id>/mainmenu.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Agenda

- Introduction
- Key concepts
- ***Developing a JSF application***
- JSF inside of an IDE
- Developing custom components
- Extending JSF
- Wrap up and book giveaway

Application requirements

- Standard web container (Servlets 2.3/Portlets 1.0 or higher, JSP 1.2 or higher)
- JSF implementation (Sun reference implementation or MyFaces/Smile open source)
- Standard Java web application
- Deployment descriptor (web.xml) must configure JSF front controller servlet:

```
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>
    javax.faces.webapp.FacesServlet
  </servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

Developing an application

- Application development steps
 - Setup web application environment
 - Edit deployment descriptor (web.xml)
 - Develop user interface screens (JSPs, templates or Java code)
 - Write backing beans
 - Add beans to configuration file
 - Integrate backing beans with user interface
- Tools will automate much of this process
- You can also develop the backing beans first, or develop both the user interface and the beans at the same time

Hello, world! demo

Agenda

- Introduction
- Key concepts
- Developing a JSF application
- ***JSF inside of an IDE***
- Developing custom components
- Extending JSF
- Wrap up and book giveaway

Agenda

- Introduction
- Key concepts
- Developing a JSF application
- JSF inside of an IDE
- ***Developing custom components***
- Extending JSF
- Wrap up and book giveaway

Developing custom components

- Generic UI functionality (calendar)
- Domain-specific functionality (report viewer, user editor)
- Develop a custom component when:
 - You need a re-usable piece of user interface functionality that doesn't currently exist on the market
- Don't develop a component when:
 - You need to add validation to an existing component (use a validator)
 - You need to convert an object into a string for display (use a converter)
 - You need to display an existing component in a different way (write a renderer)

Developing custom components

- Component or renderer development requires knowledge of servlet/portlet API and target client markup (HTML, WML, and so on)

Developing custom components

- Component development steps
 - Subclass an existing component
 - Add properties
 - Implement encoding methods (or delegate to a renderer)
 - Implement decoding method (or delegate to a renderer)
 - Register component in JSF configuration file
 - Integrate with JSP (if so desired)
 - Write tag handler class
 - Add tag to tag library descriptor (TLD)
 - Make sure tag library is included in JSP

Agenda

- Introduction
- Key concepts
- Developing a JSF application
- JSF inside of an IDE
- Developing custom components
- ***Extending JSF***
- Wrap up and book giveaway

Extending JSF

- Pluggable extension points
 - Default action listener
 - Navigation handler
 - EL
 - Variable resolver
 - Property resolver
 - View handler

Agenda

- Introduction
- Key concepts
- Developing a JSF application
- JSF inside of an IDE
- Developing custom components
- Extending JSF
- ***Wrap up and book giveaway***

Summary

- JavaServer Faces is a standard web-based user interface framework for Java
- JSF includes:
 - Stateful user interface components
 - Automatic synchronization of components with bean properties
 - Server-side handling of client-generated events
 - Validation, type conversion, navigation, and internationalization
- Supported by industry standard tools from Sun (Java Studio Creator), IBM (WSAD), Oracle (JDeveloper) and others
- Can be used with Struts and other frameworks

Resources

- JSF Central
 - News, info, products, FAQ, and an extensive list of resources
 - <http://www.jsfcentral.com>
- Sun's JSF home page
 - <http://java.sun.com/j2ee/javaserverfaces>
- MyFaces open source implementation
 - <http://www.marinschek.com/myfaces/tiki/tiki-index.php>
- Smile open source implementation
 - Class-based (Java) pages instead of JSP
 - <http://smile.sourceforge.net/index.html>
- JSF in Action home page
 - <http://www.manning.com/catalog/view.php?book=mann>

Q&A

JavaServer Faces in Action

- What's inside:
 - Thorough explanations of what JSF is, how it works, and how it relates to existing web frameworks
 - Extensive examples of using all standard components, renderers, validators, and converters
 - Comprehensive case study
 - Application design guidelines
 - In-depth coverage of Struts integration
 - Coverage of JSTL integration
 - Working with JSP and other display technologies
 - How to create custom renderers, components, validators, and converters
 - HTTP, Servlets, Portlets, JSP, and JavaBeans

