Chapter 26: JavaServer™ Faces Web Applications, Part 1

Internet & World Wide Web
How to Program, 5/e

Note: This chapter is a copy of Chapter 29 of our book *Java How to Program, 9/e*. For that reason, we simply copied the PowerPoint slides for this chapter and *did not* re-numb er them

OBJECTIVES

In this chapter you'll learn:

- To create JavaServer Faces web apps.
- To create web apps consisting of multiple pages.
- To validate user input on a web page.
- To maintain user-specific state information throughout a web app with session tracking.

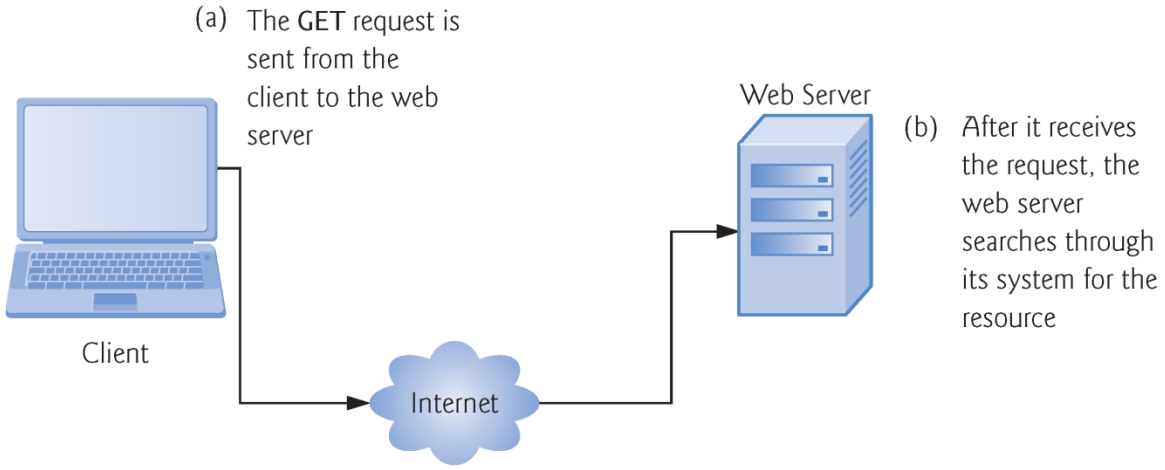


Fig. 29.1 | Client interacting with the web server. *Step 1:* The GET request.

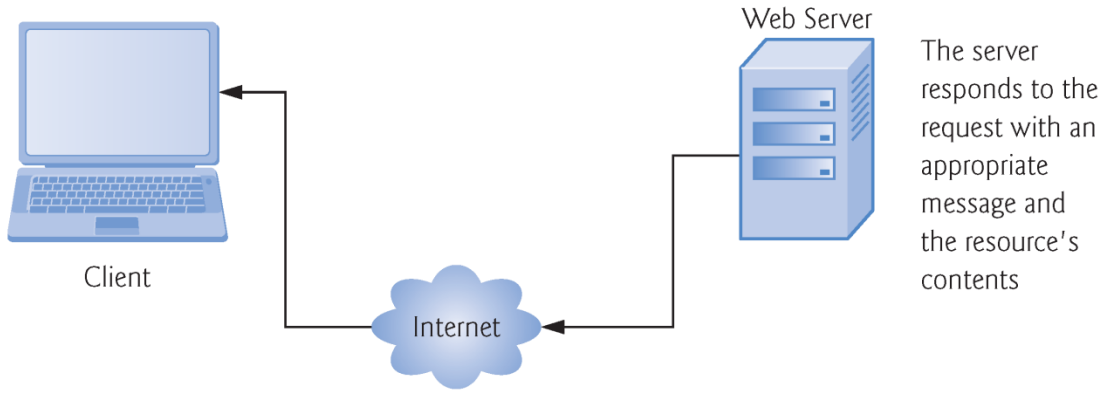


Fig. 29.2 | Client interacting with the web server. *Step 2:* The HTTP response.

Software Engineering Observation 29.1

The data sent in a POST *request is* not *part of the URL, and the user can't see the data by default. However, tools are available that expose this data, so you should not assume that the data is secure just because a* POST *request is used.*

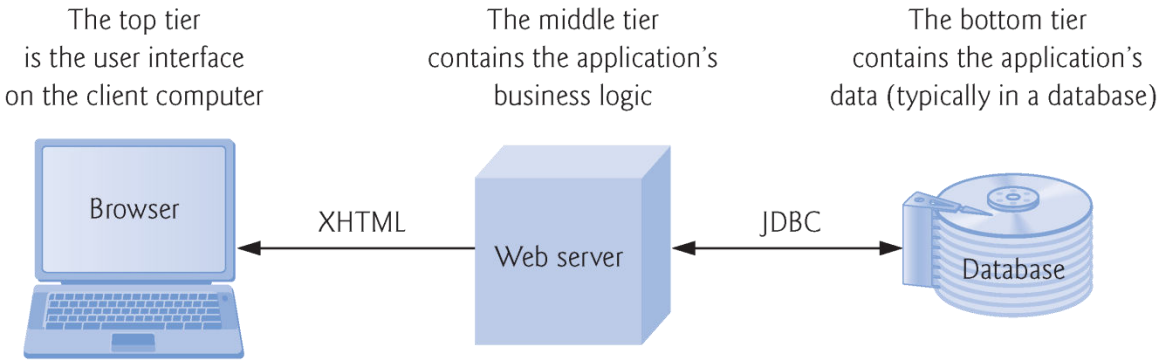


Fig. 29.3 | Three-tier architecture.

Fig. 29.4 | Sample output of the `WebTime` app.

```
1  <?xml version='1.0' encoding='UTF-8' ?>
2
3  <!-- index.xhtml -->
4  <!-- JSF page that displays the current time on the web server -->
5  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
6     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
7  <html xmlns="http://www.w3.org/1999/xhtml"
8     xmlns:h="http://java.sun.com/jsf/html">
9     <h:head>
10       <title>WebTime: A Simple Example</title>
11       <meta http-equiv="refresh" content="60" />
12    </h:head>
13    <h:body>
14       <h1>Current time on the web server: #{webTimeBean.time}</h1>
15    </h:body>
16 </html>
```

Fig. 29.5 | JSF page that displays the current time on the web server.

```
1  // WebTimeBean.java
2  // Bean that enables the JSF page to retrieve the time from the server
3  package webtime;
4
5  import java.text.DateFormat;
6  import java.util.Date;
7  import javax.faces.bean.ManagedBean;
8
9  @ManagedBean( name="webTimeBean" )
10 public class WebTimeBean
11 {
12    // return the time on the server at which the request was received
13    public String getTime()
14    {
15       return DateFormat.getTimeInstance( DateFormat.LONG ).format(
16          new Date() );
17    } // end method getTime
18 } // end class WebTimeBean
```

Fig. 29.6 | Bean that enables the JSF page to retrieve the time from the server.

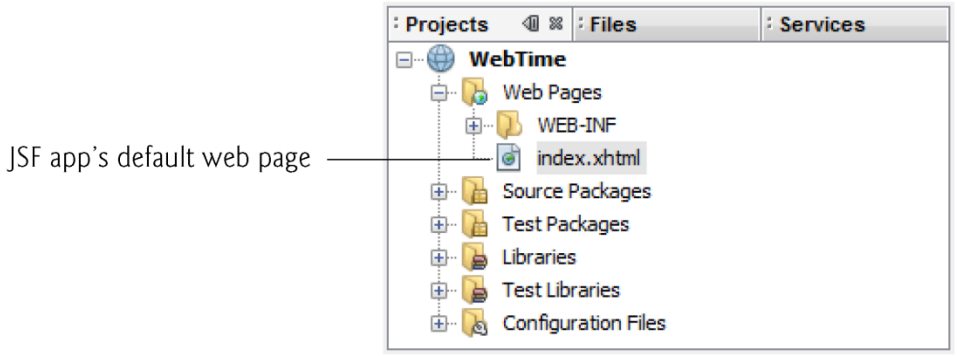


Fig. 29.7 | **Projects** window for the `WebTime` project.

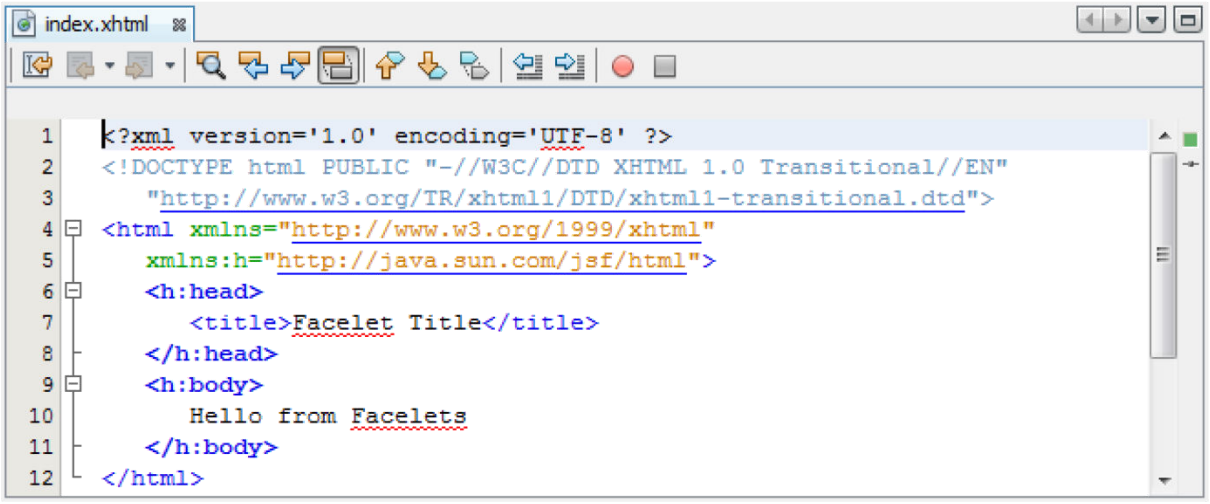


Fig. 29.8 | Default `index.xhtml` page generated by NetBeans for the web app.

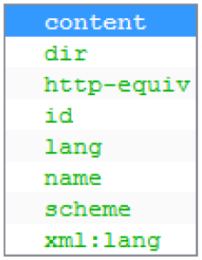


Fig. 29.9 | NetBeans code-completion window.

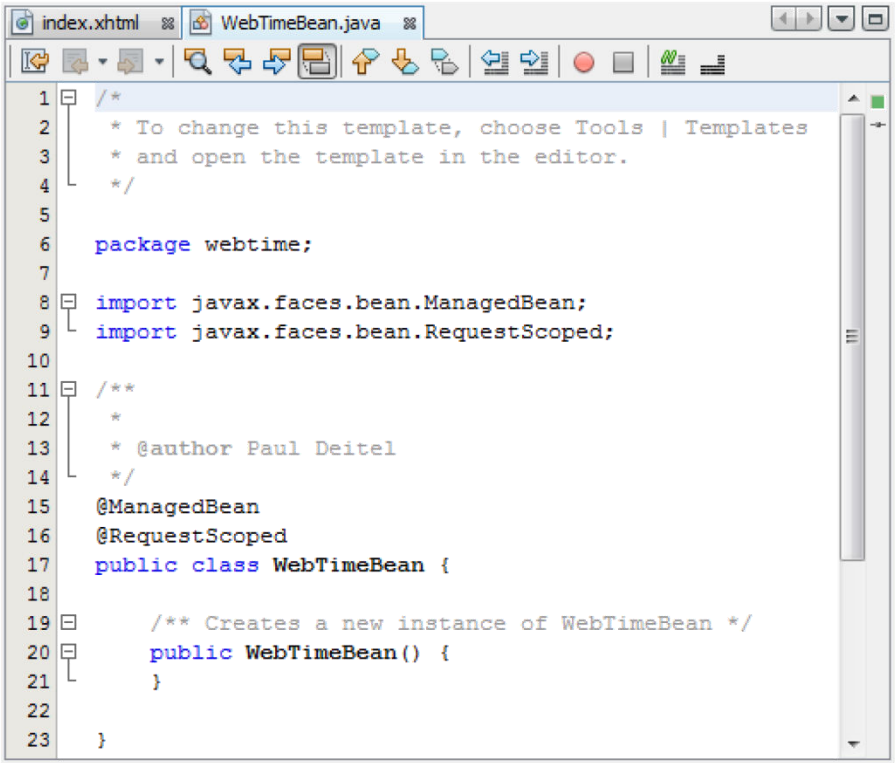


Fig. 29.10 | Default source code for the `WebTimeBean` class.

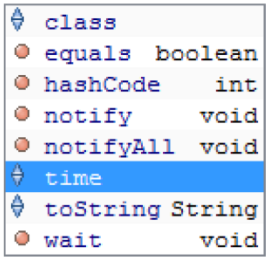


Fig. 29.11 | NetBeans code-completion window for the `webTimeBean` object.

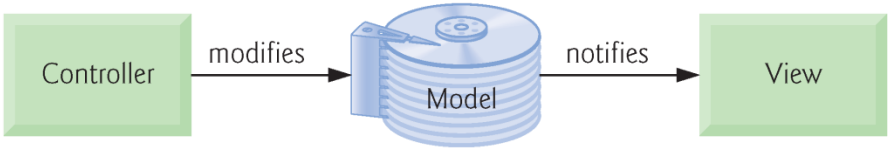


Fig. 29.12 | Model-View-Controller architecture.

JSF component	Description
`h:form`	Inserts an XHTML `form` element into a page.
`h:commandButton`	Displays a button that triggers an event when clicked. Typically, such a button is used to submit a form's user input to the server for processing.
`h:graphicImage`	Displays an image (e.g., GIF and JPG).
`h:inputText`	Displays a text box in which the user can enter input.
`h:outputLink`	Displays a hyperlink.
`h:panelGrid`	Displays an XHTML `table` element.
`h:selectOneMenu`	Displays a drop-down list of choices from which the user can make a selection.
`h:selectOneRadio`	Displays a set of radio buttons.
`f:selectItem`	Specifies an item in an `h:selectOneMenu` or `h:selectOneRadio` (and other similar components).

Fig. 29.13 | Commonly used JSF components.

```
1  <?xml version='1.0' encoding='UTF-8' ?>
2
3  <!-- index.xhtml -->
4  <!-- Registration form that demonstrates various JSF components -->
5  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
6     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
7  <html xmlns="http://www.w3.org/1999/xhtml"
8        xmlns:h="http://java.sun.com/jsf/html"
9        xmlns:f="http://java.sun.com/jsf/core">
10    <h:head>
11       <title>Sample Registration Form</title>
12    </h:head>
13    <h:body>
14       <h:form>
15          <h1>Registration Form</h1>
16          <p>Please fill in all fields and click Register</p>
17          <h:panelGrid columns="4" style="height: 96px; width:456px;">
18             <h:graphicImage name="fname.png" library="images"/>
19             <h:inputText id="firstNameInputText"/>
20             <h:graphicImage name="lname.png" library="images"/>
21             <h:inputText id="lastNameInputText"/>
22             <h:graphicImage name="email.png" library="images"/>
```

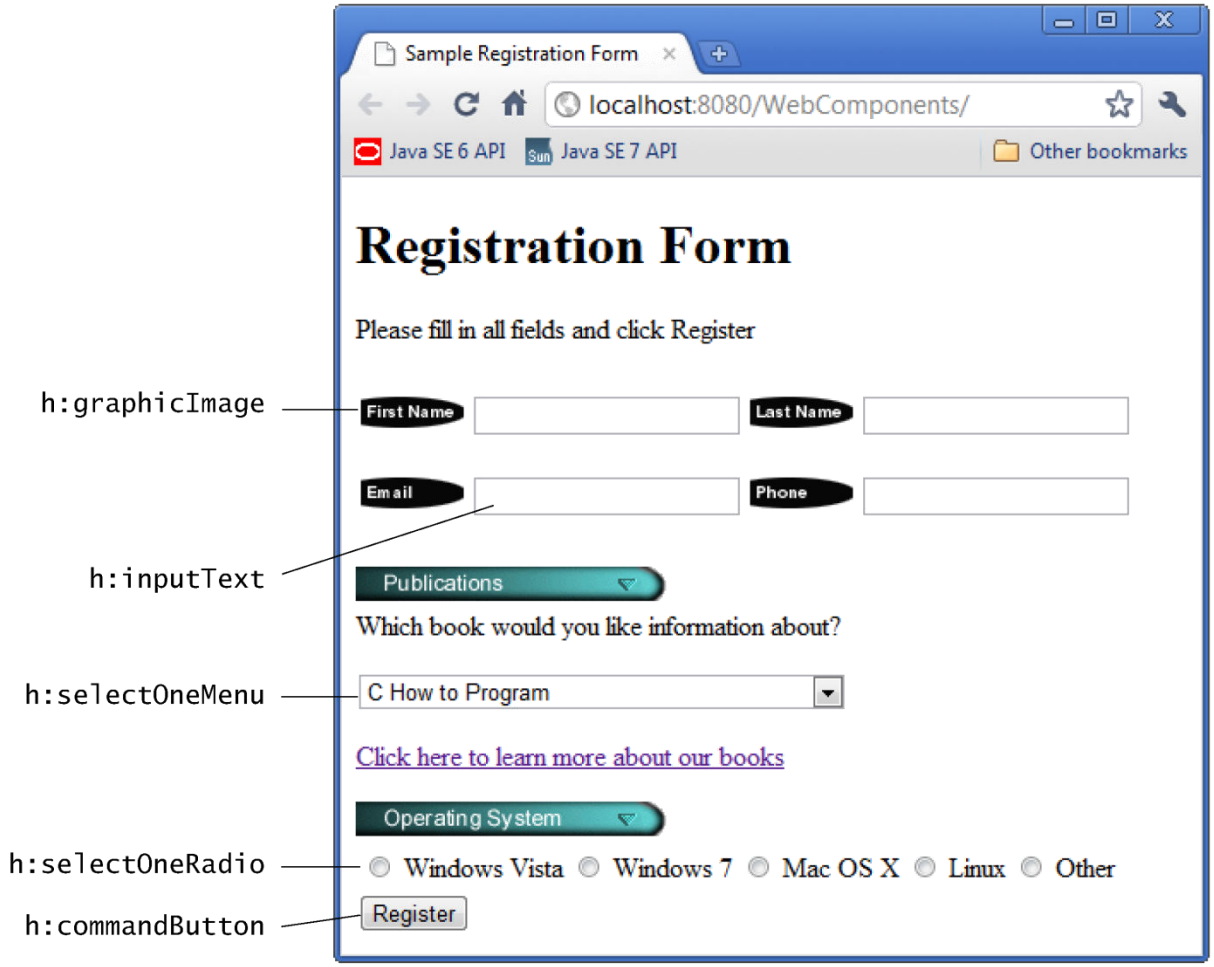
Fig. 29.14 | Registration form that demonstrates various JSF components. (Part 1 of 4.)

```
23              <h:inputText id="emailInputText"/>
24              <h:graphicImage name="phone.png" library="images"/>
25              <h:inputText id="phoneInputText"/>
26           </h:panelGrid>
27           <p><h:graphicImage name="publications.png" library="images"/>
28              <br/>Which book would you like information about?</p>
29           <h:selectOneMenu id="booksSelectOneMenu">
30              <f:selectItem itemValue="CHTP"
31                 itemLabel="C How to Program" />
32              <f:selectItem itemValue="CPPHTP"
33                 itemLabel="C++ How to Program" />
34              <f:selectItem itemValue="IW3HTP"
35                 itemLabel="Internet &amp;amp; World Wide Web How to Program" />
36              <f:selectItem itemValue="JHTP"
37                 itemLabel="Java How to Program" />
38              <f:selectItem itemValue="VBHTP"
39                 itemLabel="Visual Basic How to Program" />
40              <f:selectItem itemValue="VCSHTP"
41                 itemLabel="Visual C# How to Program" />
42           </h:selectOneMenu>
43           <p><h:outputLink value="http://www.deitel.com">
44              Click here to learn more about our books
```

Fig. 29.14 | Registration form that demonstrates various JSF components. (Part 2 of 4.)

```
45            </h:outputLink></p>
46            <h:graphicImage name="os.png" library="images"/>
47            <h:selectOneRadio id="osSelectOneRadio">
48               <f:selectItem itemValue="WinVista" itemLabel="Windows Vista"/>
49               <f:selectItem itemValue="Win7" itemLabel="Windows 7"/>
50               <f:selectItem itemValue="OSX" itemLabel="Mac OS X"/>
51               <f:selectItem itemValue="Linux" itemLabel="Linux"/>
52               <f:selectItem itemValue="Other" itemLabel="Other"/>
53            </h:selectOneRadio>
54            <h:commandButton value="Register"/>
55         </h:form>
56      </h:body>
57   </html>
```

Fig. 29.14 | Registration form that demonstrates various JSF components. (Part 3 of 4.)



```
1  // ValidationBean.java
2  // Validating user input.
3  package validation;
4
5  import java.io.Serializable;
6  import javax.faces.bean.ManagedBean;
7
8  @ManagedBean( name="validationBean" )
9  public class ValidationBean implements Serializable
10 {
11    private String name;
12    private String email;
13    private String phone;
14
15    // return the name String
16    public String getName()
17    {
18       return name;
19    } // end method getName
20
```

Fig. 29.15 | `ValidationBean` stores the validated data, which is then used as part of the response to the client. (Part 1 of 3.)

```
21      // set the name String
22      public void setName( String name )
23      {
24         this.name = name;
25      } // end method setName
26
27      // return the email String
28      public String getEmail()
29      {
30         return email;
31      } // end method getEmail
32
33      // set the email String
34      public void setEmail( String email )
35      {
36         this.email = email;
37      } // end method setEmail
38
```

Fig. 29.15 | `ValidationBean` stores the validated data, which is then used as part of the response to the client. (Part 2 of 3.)

```
39      // return the phone String
40      public String getPhone()
41      {
42         return phone;
43      } // end method getPhone
44
45      // set the phone String
46      public void setPhone( String phone )
47      {
48         this.phone = phone;
49      } // end method setPhone
50
51      // returns result for rendering on the client
52      public String getResult()
53      {
54         if ( name != null && email != null && phone != null )
55            return "<p style=\"background-color:yellow;width:200px;" +
56               "padding:5px\">Name: " + getName() + "<br/>E-Mail: " +
57               getEmail() + "<br/>Phone: " + getPhone() + "</p>";
58         else
59            return ""; // request has not yet been made
60      } // end method getResult
61   } // end class ValidationBean
```

Fig. 29.15 | `ValidationBean` stores the validated data, which is then used as part of the response to the client. (Part 3 of 3.)

```
1  <?xml version='1.0' encoding='UTF-8' ?>
2
3  <!-- index.xhtml -->
4  <!-- Validating user input -->
5  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
6     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
7  <html xmlns="http://www.w3.org/1999/xhtml"
8     xmlns:h="http://java.sun.com/jsf/html"
9     xmlns:f="http://java.sun.com/jsf/core">
10    <h:head>
11       <title>Validating Form Data</title>
12       <h:outputStylesheet name="style.css" library="css"/>
13    </h:head>
14    <h:body>
15       <h:form>
16          <h1>Please fill out the following form:</h1>
17          <p>All fields are required and must contain valid information</p>
18          <h:panelGrid columns="3">
19             <h:outputText value="Name:"/>
20             <h:inputText id="nameInputText" required="true"
21                requiredMessage="Please enter your name"
22                value="#{validationBean.name}"
```

Fig. 29.16 | Form to demonstrate validating user input. (Part 1 of 7.)

```
23                validatorMessage="Name must be fewer than 30 characters">
24                <f:validateLength maximum="30" />
25             </h:inputText>
26             <h:message for="nameInputText" styleClass="error"/>
27             <h:outputText value="E-mail:"/>
28             <h:inputText id="emailInputText" required="true"
29                requiredMessage="Please enter a valid e-mail address"
30                value="#{validationBean.email}"
31                validatorMessage="Invalid e-mail address format">
32                <f:validateRegex pattern=
33                   "\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*" />
34             </h:inputText>
35             <h:message for="emailInputText" styleClass="error"/>
36             <h:outputText value="Phone:"/>
37             <h:inputText id="phoneInputText" required="true"
38                requiredMessage="Please enter a valid phone number"
39                value="#{validationBean.phone}"
40                validatorMessage="Invalid phone number format">
41                <f:validateRegex pattern=
42                   "((\(\d{3}\) ?)|(\d{3}-))?\d{3}-\d{4}" />
43             </h:inputText>
44             <h:message for="phoneInputText" styleClass="error"/>
```

Fig. 29.16 | Form to demonstrate validating user input. (Part 2 of 7.)

```
45            </h:panelGrid>
46            <h:commandButton value="Submit"/>
47            <h:outputText escape="false" value="#{validationBean.response}"/>
48         </h:form>
49      </h:body>
50   </html>
```

Fig. 29.16 | Form to demonstrate validating user input. (Part 3 of 7.)

a) Submitting the form before entering any information

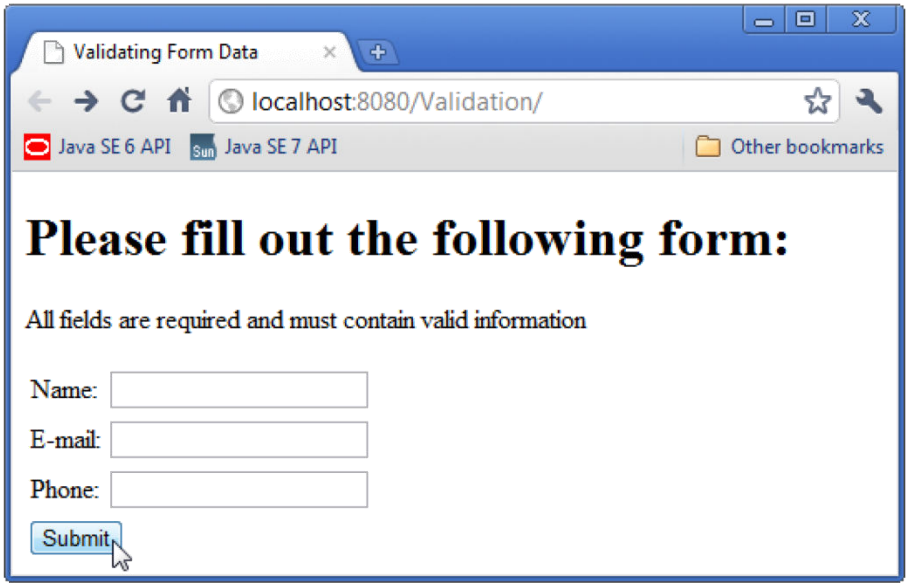


Fig. 29.16 | Form to demonstrate validating user input. (Part 4 of 7.)

b) Error messages displayed after submitting the empty form

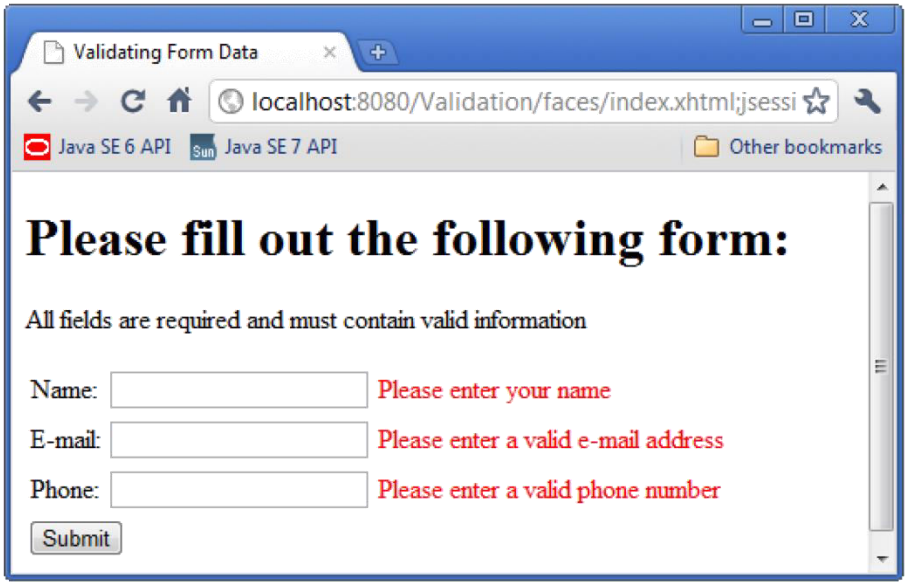


Fig. 29.16 | Form to demonstrate validating user input. (Part 5 of 7.)

c) Error messages displayed after submitting invalid information

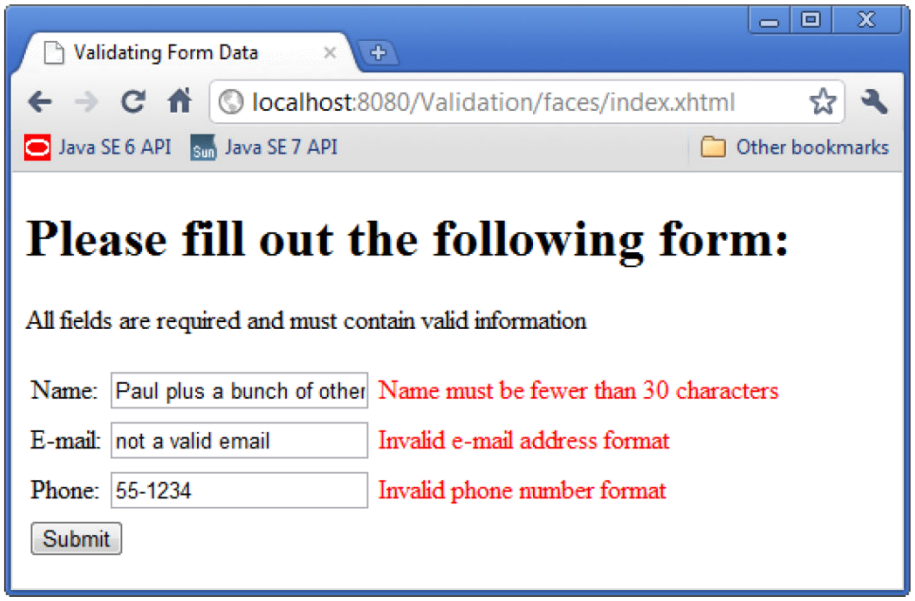


Fig. 29.16 | Form to demonstrate validating user input. (Part 6 of 7.)

d) Successfully submitted form

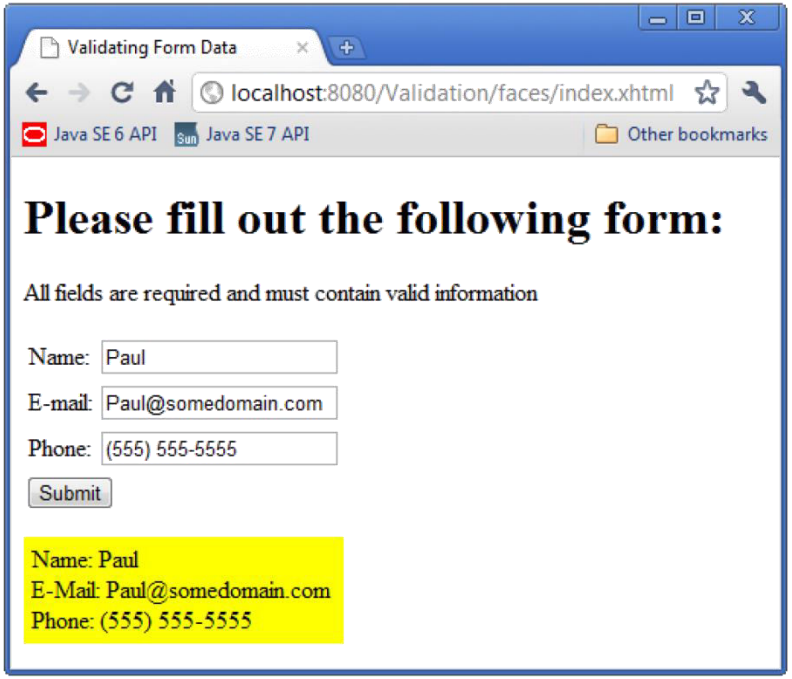


Fig. 29.16 | Form to demonstrate validating user input. (Part 7 of 7.)

Portability Tip 29.1

Users may disable cookies in their web browsers to help ensure their privacy. Such users will experience difficulty using web applications that depend on cookies to maintain state information.

Software Engineering Observation 29.2

`@SessionScoped` *beans should implement the* `Serializable` *interface. Websites with heavy traffic often use groups of servers (sometimes hundreds or thousands of them) to respond to requests. Such groups are known as server farms. Server farms often balance the number of requests being handled on each server by moving some sessions to other servers. Making a bean* `Serializable` *enables the session to be moved properly among servers.*

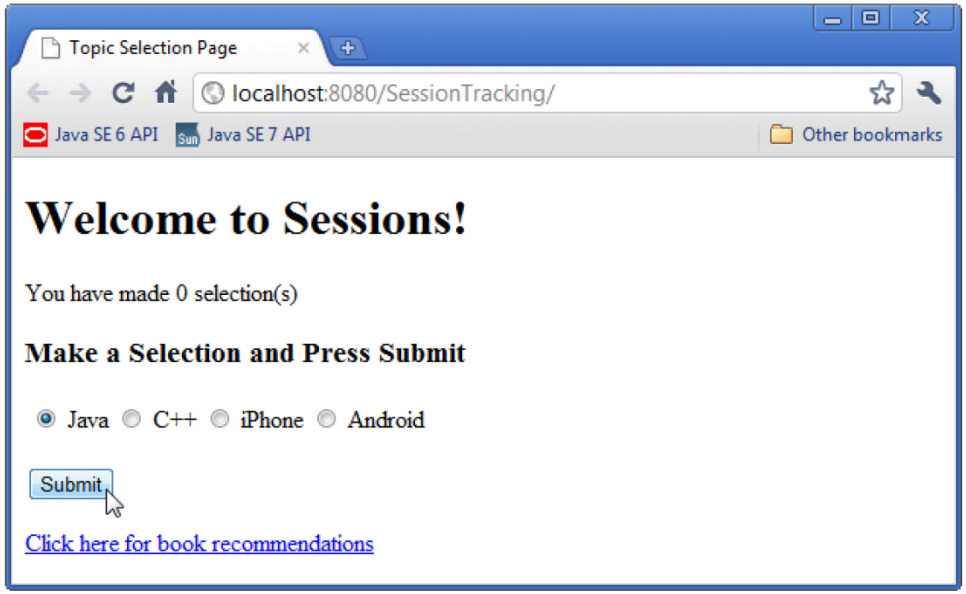


Fig. 29.17 | `index.xhtml` after the user has made a selection and is about to submit the form for the first time.

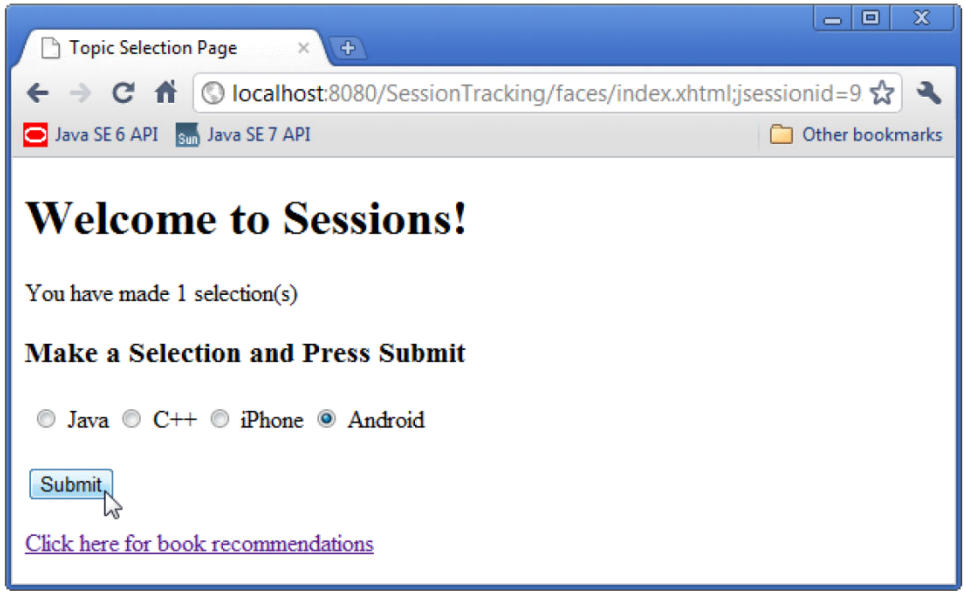


Fig. 29.18 | `index.xhtml` after the user has submitted the form the first time, made another selection and is about to submit the form again.

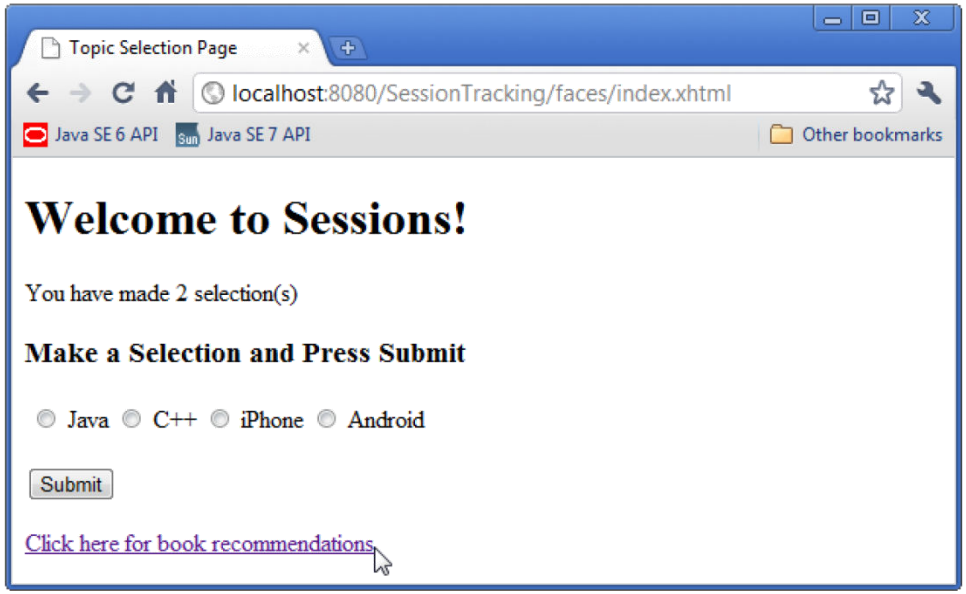


Fig. 29.19 | `index.xhtml` after the user has submitted the form the second time and is about to click the link to the `recommendations.xhtml` page.

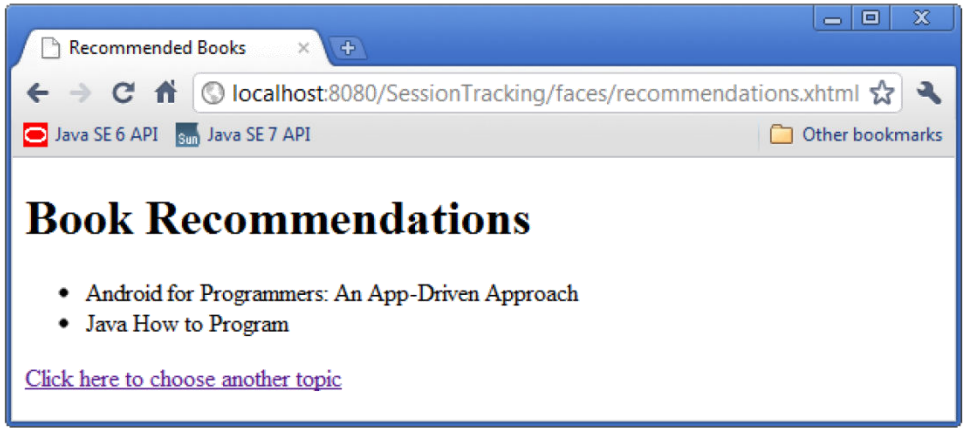


Fig. 29.20 | `recommendations.hxtml` showing book recommendations for the topic selections made by the user in Figs. 29.18 and 29.19.

```
1  // SelectionsBean.java
2  // Manages a user's topic selections
3  package sessiontracking;
4
5  import java.io.Serializable;
6  import java.util.HashMap;
7  import java.util.Set;
8  import java.util.TreeSet;
9  import javax.faces.bean.ManagedBean;
10 import javax.faces.bean.SessionScoped;
11
12 @ManagedBean( name="selectionsBean" )
13 @SessionScoped
14 public class SelectionsBean implements Serializable
15 {
16    // map of topics to book titles
17    private static final HashMap< String, String > booksMap =
18       new HashMap< String, String >();
19
```

Fig. 29.21 | `@SessionScoped SelectionsBean` class. (Part 1 of 4.)

```
20      // initialize booksMap
21      static
22      {
23         booksMap.put( "java", "Java How to Program" );
24         booksMap.put( "cpp", "C++ How to Program" );
25         booksMap.put( "iphone",
26            "iPhone for Programmers: An App-Driven Approach" );
27         booksMap.put( "android",
28            "Android for Programmers: An App-Driven Approach" );
29      } // end static initializer block
30
31      // stores individual user's selections
32      private Set< String > selections = new TreeSet< String >();
33      private String selection; // stores the current selection
34
```

Fig. 29.21 | `@SessionScoped SelectionsBean` class. (Part 2 of 4.)

```
35      // return number of selections
36      public int getNumberOfSelections()
37      {
38         return selections.size();
39      } // end method getNumberOfSelections
40
41      // returns the current selection
42      public String getSelection()
43      {
44         return selection;
45      } // end method getSelection
46
47      // store user's selection
48      public void setSelection( String topic )
49      {
50         selection = booksMap.get( topic );
51         selections.add( selection );
52      } // end method setSelection
53
```

Fig. 29.21 | `@SessionScoped SelectionsBean` class. (Part 3 of 4.)

```
54      // return the Set of selections
55      public String[] getSelections()
56      {
57         return selections.toArray( new String[ selections.size() ] );
58      } // end method getSelections
59   } // end class SelectionsBean
```

Fig. 29.21 | `@SessionScoped SelectionsBean` class. (Part 4 of 4.)

```
1  <?xml version='1.0' encoding='UTF-8' ?>
2
3  <!-- index.xhtml -->
4  <!-- Allow the user to select a topic -->
5  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
6     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
7  <html xmlns="http://www.w3.org/1999/xhtml"
8     xmlns:h="http://java.sun.com/jsf/html"
9     xmlns:f="http://java.sun.com/jsf/core">
10    <h:head>
11       <title>Topic Selection Page</title>
12    </h:head>
13    <h:body>
14       <h1>Welcome to Sessions!</h1>
15       <p>You have made #{selectionsBean.numberOfSelections} selection(s)
16       </p>
17       <h3>Make a Selection and Press Submit</h3>
18       <h:form>
19          <h:selectOneRadio id="topicSelectOneRadio" required="true"
20             requiredMessage="Please choose a topic, then press Submit"
21             value="#{selectionsBean.selection}">
22             <f:selectItem itemValue="java" itemLabel="Java"/>
23             <f:selectItem itemValue="cpp" itemLabel="C++"/>
```

Fig. 29.22 | `index.xhtml` allows the user to select a topic. (Part 1 of 2.)

```
24              <f:selectItem itemValue="iphone" itemLabel="iPhone"/>
25              <f:selectItem itemValue="android" itemLabel="Android"/>
26           </h:selectOneRadio>
27           <p><h:commandButton value="Submit"/></p>
28        </h:form>
29        <p><h:outputLink value="recommendations.xhtml">
30           Click here for book recommendations
31        </h:outputLink></p>
32     </h:body>
33  </html>
```

Fig. 29.22 | `index.xhtml` allows the user to select a topic. (Part 2 of 2.)

```
1  <?xml version='1.0' encoding='UTF-8' ?>
2
3  <!-- recommendations.xhtml -->
4  <!-- Display recommended books based on the user's selected topics -->
5  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
6     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
7  <html xmlns="http://www.w3.org/1999/xhtml"
8     xmlns:h="http://java.sun.com/jsf/html"
9     xmlns:ui="http://java.sun.com/jsf/facelets">
10    <h:head>
11       <title>Recommended Books</title>
12    </h:head>
13    <h:body>
14       <h1>Book Recommendations</h1>
15       <ul>
16          <ui:repeat value="#{selectionsBean.selections}" var="book">
17             <li>#{book}</li>
18          </ui:repeat>
19       </ul>
20       <p><h:outputLink value="index.xhtml">
21             Click here to choose another topic
22          </h:outputLink></p>
23    </h:body>
24 </html>
```

Fig. 29.23 | `recommendations.xhtml` displays book recommendations based on the user's selections.