Chapter 30 JavaServer™ Faces Web Applications, Part 2

Java How to Program, 9/e

Chapter 27: JavaServer™ Faces Web Applications, Part 2

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

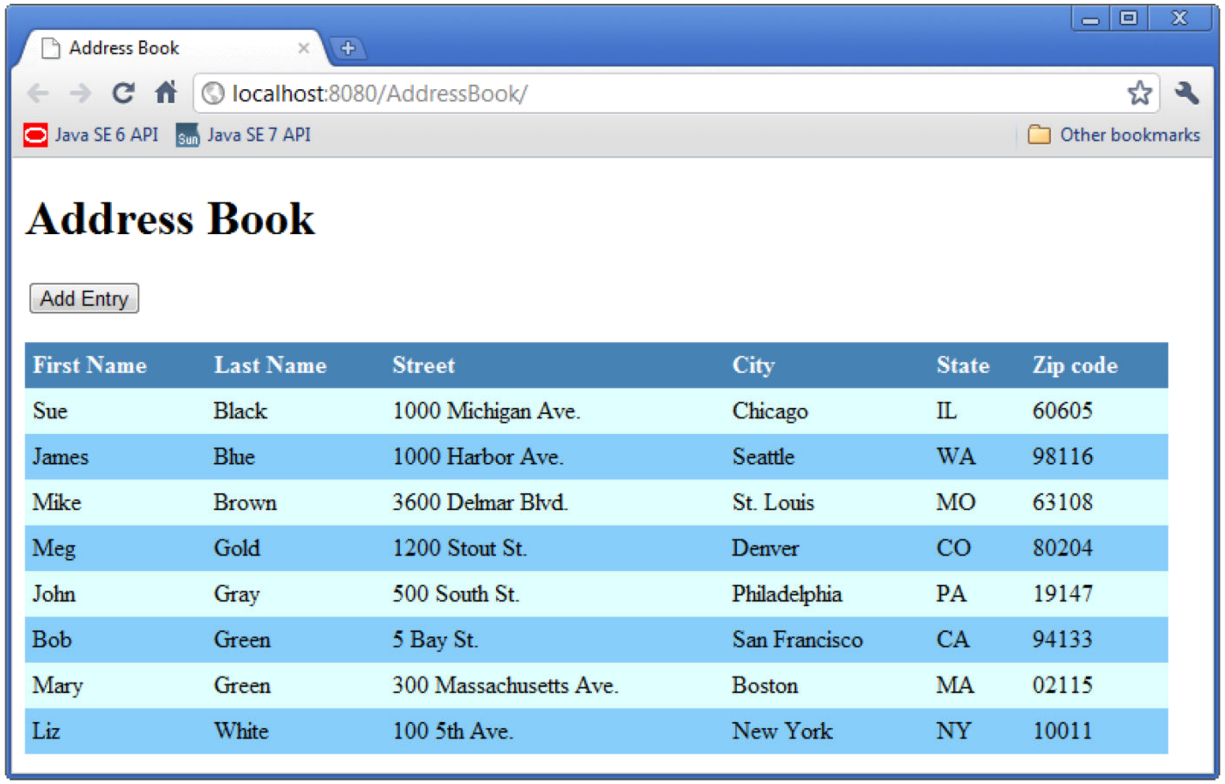
Note: This chapter is a copy of Chapter 30 of our book *Java How to Program, 9/e*. For that reason, we simply copied the PowerPoint slides for this chapter and *did not* re-numb er them

OBJECTIVES

In this chapter you'll learn:

- To access databases from JSF applications.
- The basic principles and advantages of Ajax technology.
- To use Ajax in a JSF web app.

a) Table of addresses displayed when the AddressBook app is first requested



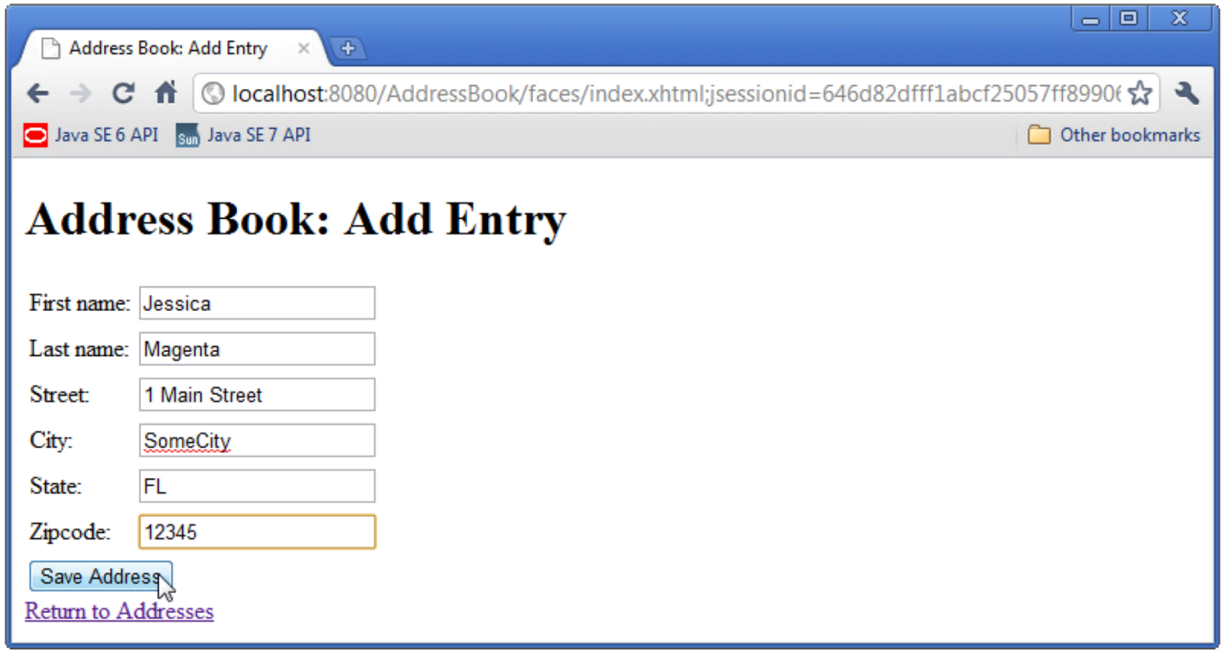
Address Book

Add Entry

First Name	Last Name	Street	City	State	Zip code
Sue	Black	1000 Michigan Ave.	Chicago	IL	60605
James	Blue	1000 Harbor Ave.	Seattle	WA	98116
Mike	Brown	3600 Delmar Blvd.	St. Louis	MO	63108
Meg	Gold	1200 Stout St.	Denver	CO	80204
John	Gray	500 South St.	Philadelphia	PA	19147
Bob	Green	5 Bay St.	San Francisco	CA	94133
Mary	Green	300 Massachusetts Ave.	Boston	MA	02115
Liz	White	100 5th Ave.	New York	NY	10011

Fig. 30.1 | Sample outputs from the AddressBook app. (Part 1 of 3.)

b) Form for adding an entry


Address Book: Add Entry
localhost:8080/AddressBook/faces/index.xhtml;jsessionid=646d82dfff1abcf25057ff8990
Java SE 6 API
Java SE 7 API
Other bookmarks

Address Book: Add Entry

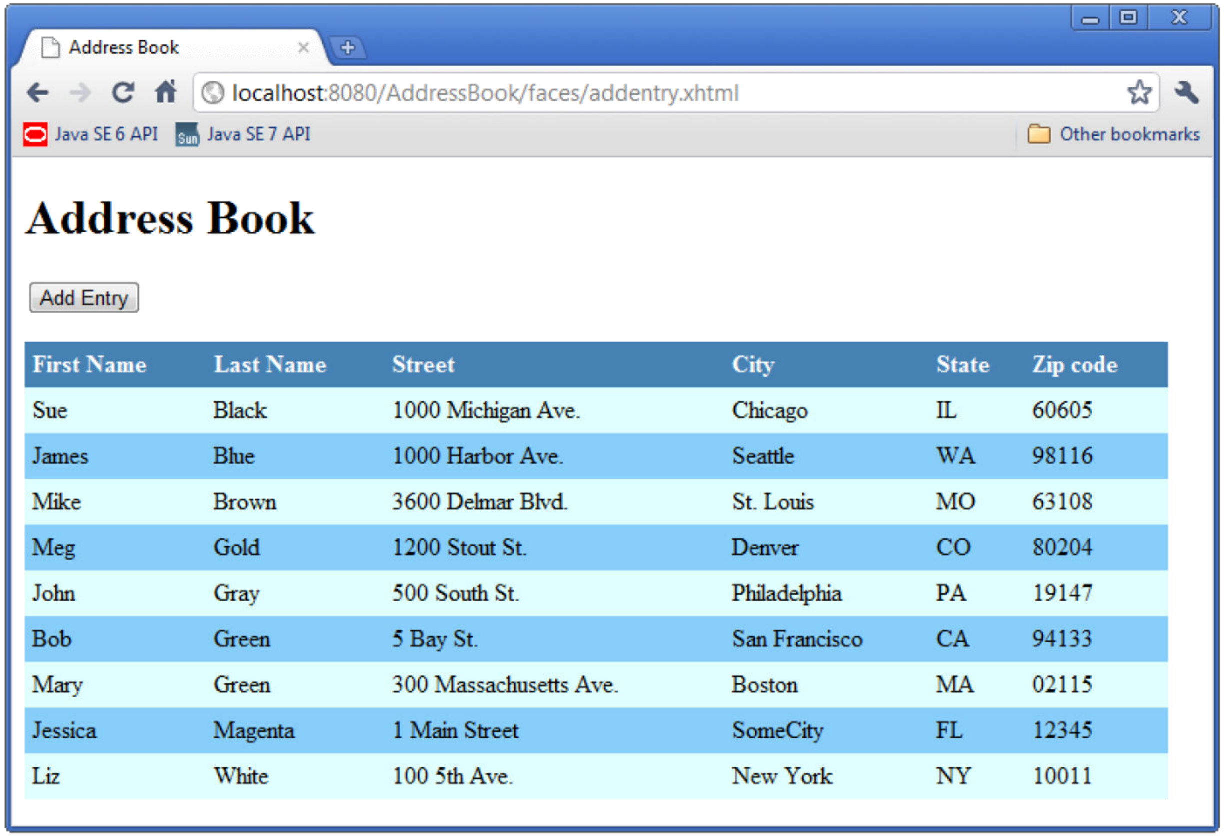
First name:	Jessica
Last name:	Magenta
Street:	1 Main Street
City:	SomeCity
State:	FL
Zipcode:	12345

Save Address

Return to Addresses

Fig. 30.1 | Sample outputs from the `AddressBook` app. (Part 2 of 3.)

c) Table of addresses updated with the new entry added in Part (b)



Address Book

First Name	Last Name	Street	City	State	Zip code
Sue	Black	1000 Michigan Ave.	Chicago	IL	60605
James	Blue	1000 Harbor Ave.	Seattle	WA	98116
Mike	Brown	3600 Delmar Blvd.	St. Louis	MO	63108
Meg	Gold	1200 Stout St.	Denver	CO	80204
John	Gray	500 South St.	Philadelphia	PA	19147
Bob	Green	5 Bay St.	San Francisco	CA	94133
Mary	Green	300 Massachusetts Ave.	Boston	MA	02115
Jessica	Magenta	1 Main Street	SomeCity	FL	12345
Liz	White	100 5th Ave.	New York	NY	10011

Fig. 30.1 | Sample outputs from the `AddressBook` app. (Part 3 of 3.)

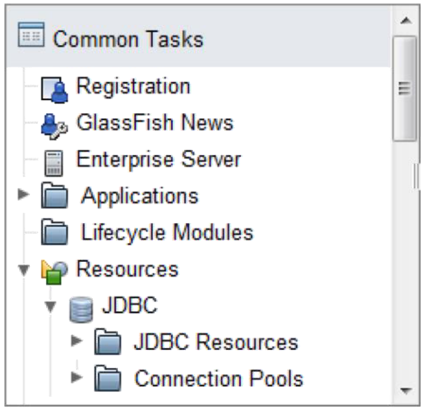
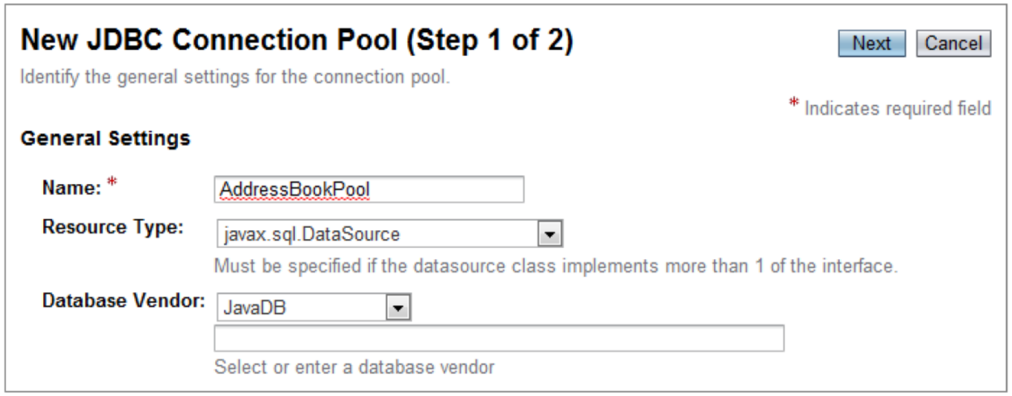


Fig. 30.2 | **Common Tasks** window in the GlassFish server configuration web page.



New JDBC Connection Pool (Step 1 of 2)

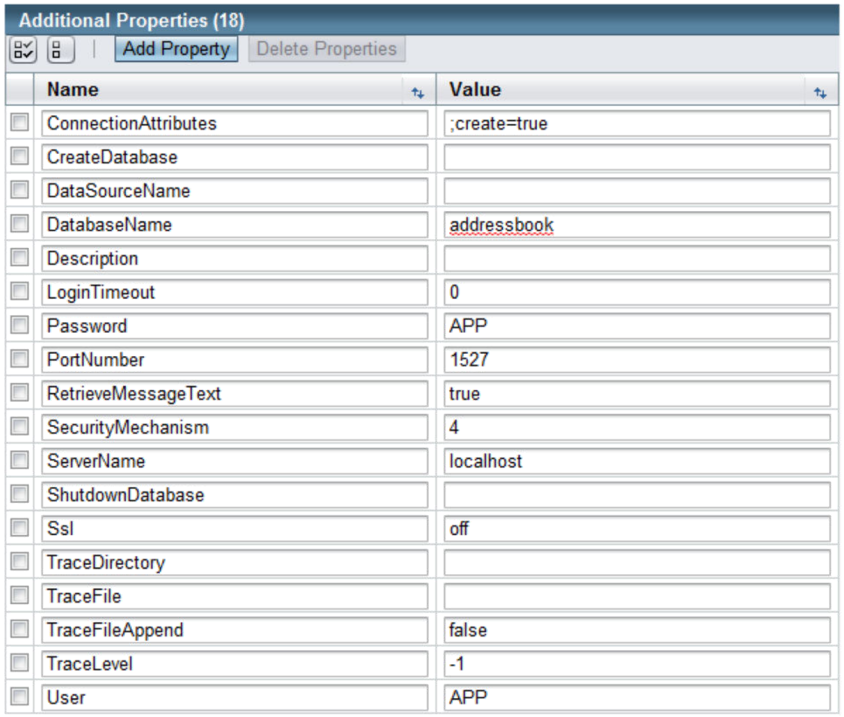
Identify the general settings for the connection pool.

* Indicates required field

General Settings

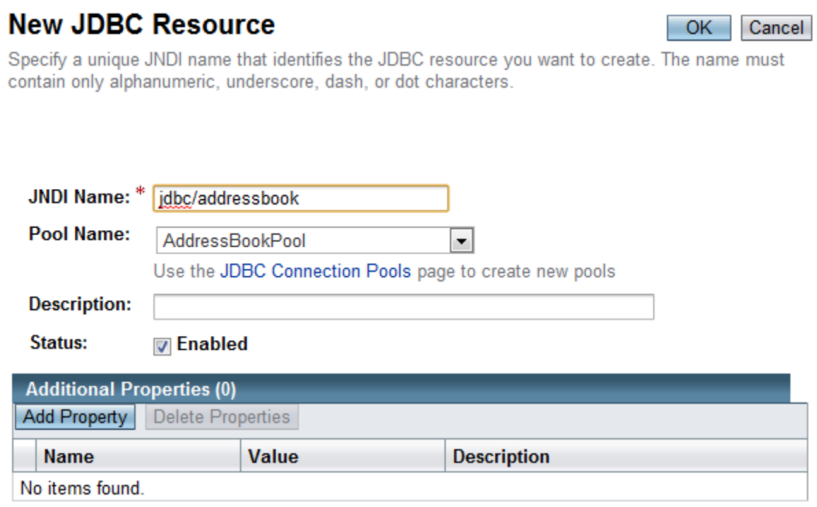
Name: *	AddressBookPool
Resource Type:	javax.sql.DataSource Must be specified if the datasource class implements more than 1 of the interface.
Database Vendor:	JavaDB Select or enter a database vendor

Fig. 30.3 | New JDBC Connection Pool (Step 1 of 2) page.



Name	Value
ConnectionAttributes	;create=true
CreateDatabase	
DataSourceName	
DatabaseName	addressbook
Description	
LoginTimeout	0
Password	APP
PortNumber	1527
RetrieveMessageText	true
SecurityMechanism	4
ServerName	localhost
ShutdownDatabase	
Ssl	off
TraceDirectory	
TraceFile	
TraceFileAppend	false
TraceLevel	-1
User	APP

Fig. 30.4 | New JDBC Connection Pool (Step 2 of 2) page.


New JDBC Resource

Specify a unique JNDI name that identifies the JDBC resource you want to create. The name must contain only alphanumeric, underscore, dash, or dot characters.

JNDI Name: *	jdbc/addressbook
Pool Name:	AddressBookPool
	Use the JDBC Connection Pools page to create new pools
Description:	
Status:	✓ Enabled

Additional Properties (0)

	Name	Value	Description
No items found.			

Fig. 30.5 | New JDBC Resource page.

```java
// AddressBean.java
// Bean for interacting with the AddressBook database
package addressbook;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.annotation.Resource;
import javax.faces.bean.ManagedBean;
import javax.sql.DataSource;
import javax.sql.rowset.CachedRowSet;

@ManagedBean( name="addressBean" )
public class AddressBean
{
```

Fig. 30.6 | `AddressBean` interacts with a database to store and retrieve addresses. (Part 1 of 10.)

```
   // instance variables that represent one address
   private String firstName;
   private String lastName;
   private String street;
   private String city;
   private String state;
   private String zipcode;

   // allow the server to inject the DataSource
   @Resource( name="jdbc/addressbook" )
   DataSource dataSource;

   // get the first name
   public String getFirstName()
   {
      return firstName;
   } // end method getFirstName

```

Fig. 30.6 | `AddressBean` interacts with a database to store and retrieve addresses. (Part 2 of 10.)

```
   // set the first name
   public void setFirstName( String firstName )
   {
      this.firstName = firstName;
   } // end method setFirstName

   // get the last name
   public String getLastName()
   {
      return lastName;
   } // end method getLastName

   // set the last name
   public void setLastName( String lastName )
   {
      this.lastName = lastName;
   } // end method setLastName

   // get the street
   public String getStreet()
   {
      return street;
   } // end method getStreet
```

Fig. 30.6 | `AddressBean` interacts with a database to store and retrieve addresses. (Part 3 of 10.)

```java

   // set the street
   public void setStreet( String street )
   {
      this.street = street;
   } // end method setStreet

   // get the city
   public String getCity()
   {
      return city;
   } // end method getCity

   // set the city
   public void setCity( String city )
   {
      this.city = city;
   } // end method setCity

```

Fig. 30.6 | `AddressBean` interacts with a database to store and retrieve addresses. (Part 4 of 10.)

```
   // get the state
   public String getState()
   {
      return state;
   } // end method getState

   // set the state
   public void setState( String state )
   {
      this.state = state;
   } // end method setState

   // get the zipcode
   public String getZipcode()
   {
      return zipcode;
   } // end method getZipcode

```

Fig. 30.6 | `AddressBean` interacts with a database to store and retrieve addresses. (Part 5 of 10.)

```java
   // set the zipcode
   public void setZipcode( String zipcode )
   {
      this.zipcode = zipcode;
   } // end method setZipcode

   // return a ResultSet of entries
   public ResultSet getAddresses() throws SQLException
   {
      // check whether dataSource was injected by the server
      if ( dataSource == null )
         throw new SQLException( "Unable to obtain DataSource" );

      // obtain a connection from the connection pool
      Connection connection = dataSource.getConnection();

```

Fig. 30.6 | `AddressBean` interacts with a database to store and retrieve addresses. (Part 6 of 10.)

```
      // check whether connection was successful
      if ( connection == null )
         throw new SQLException( "Unable to connect to DataSource" );

      try
      {
         // create a PreparedStatement to insert a new address book entry
         PreparedStatement getAddresses = connection.prepareStatement(
            "SELECT FIRSTNAME, LASTNAME, STREET, CITY, STATE, ZIP " +
            "FROM ADDRESSES ORDER BY LASTNAME, FIRSTNAME" );

         CachedRowSet rowSet = new com.sun.rowset.CachedRowSetImpl();
         rowSet.populate( getAddresses.executeQuery() );
         return rowSet;
      } // end try
      finally
      {
         connection.close(); // return this connection to pool
      } // end finally
   } // end method getAddresses

```

Fig. 30.6 | AddressBean interacts with a database to store and retrieve addresses. (Part 7 of 10.)

```
   // save a new address book entry
   public String save() throws SQLException
   {
      // check whether dataSource was injected by the server
      if ( dataSource == null )
         throw new SQLException( "Unable to obtain DataSource" );

      // obtain a connection from the connection pool
      Connection connection = dataSource.getConnection();

      // check whether connection was successful
      if ( connection == null )
         throw new SQLException( "Unable to connect to DataSource" );

```

Fig. 30.6 | `AddressBean` interacts with a database to store and retrieve addresses. (Part 8 of 10.)

```
         try
         {
            // create a PreparedStatement to insert a new address book entry
            PreparedStatement addEntry =
               connection.prepareStatement( "INSERT INTO ADDRESSES " +
                  "(FIRSTNAME,LASTNAME,STREET,CITY,STATE,ZIP)" +
                  "VALUES ( ?, ?, ?, ?, ?, ? )" );

            // specify the PreparedStatement's arguments
            addEntry.setString( 1, getFirstName() );
            addEntry.setString( 2, getLastName() );
            addEntry.setString( 3, getStreet() );
            addEntry.setString( 4, getCity() );
            addEntry.setString( 5, getState() );
            addEntry.setString( 6, getZipcode() );

            addEntry.executeUpdate(); // insert the entry
            return "index"; // go back to index.xhtml page
         } // end try
```

Fig. 30.6 | AddressBean interacts with a database to store and retrieve addresses. (Part 9 of 10.)

```
        finally
        {
           connection.close(); // return this connection to pool
        } // end finally
     } // end method save
  } // end class AddressBean
```

Fig. 30.6 | `AddressBean` interacts with a database to store and retrieve addresses. (Part 10 of 10.)

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

<!-- index.html -->
<!-- Displays an h:dataTable of the addresses in the address book -->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:f="http://java.sun.com/jsf/core">
   <h:head>
      <title>Address Book</title>
      <h:outputStylesheet name="style.css" library="css"/>
   </h:head>
   <h:body>
      <h1>Address Book</h1>
      <h:form>
         <p><h:commandButton value="Add Entry" action="addentry"/></p>
      </h:form>
      <h:dataTable value="#{addressBean.addresses}" var="address"
         rowClasses="oddRows,evenRows" headerClass="header"
         styleClass="table" cellpadding="5" cellspacing="0">
```

Fig. 30.7 | Displays an `h:dataTable` of the addresses in the address book. (Part 1 of 3.)

```
            <h:column>
               <f:facet name="header">First Name</f:facet>
               #{address.FIRSTNAME}
            </h:column>
            <h:column>
               <f:facet name="header">Last Name</f:facet>
               #{address.LASTNAME}
            </h:column>
            <h:column>
               <f:facet name="header">Street</f:facet>
               #{address.STREET}
            </h:column>
            <h:column>
               <f:facet name="header">City</f:facet>
               #{address.CITY}
            </h:column>
            <h:column>
               <f:facet name="header">State</f:facet>
               #{address.STATE}
            </h:column>
```

Fig. 30.7 | Displays an `h:dataTable` of the addresses in the address book. (Part 2 of 3.)

```
            <h:column>
               <f:facet name="header">Zip code</f:facet>
               #{address.ZIP}
            </h:column>
         </h:dataTable>
      </h:body>
   </html>
```

Fig. 30.7 | Displays an `h:dataTable` of the addresses in the address book. (Part 3 of 3.)

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

<!-- addentry.html -->
<!-- Form for adding an entry to an address book -->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:h="http://java.sun.com/jsf/html">
   <h:head>
      <title>Address Book: Add Entry</title>
      <h:outputStylesheet name="style.css" library="css"/>
   </h:head>
   <h:body>
      <h1>Address Book: Add Entry</h1>
      <h:form>
         <h:panelGrid columns="3">
            <h:outputText value="First name:"/>
            <h:inputText id="firstNameInputText" required="true"
               requiredMessage="Please enter first name"
               value="#{addressBean.firstName}" maxlength="30"/>
            <h:message for="firstNameInputText" styleClass="error"/>
            <h:outputText value="Last name:"/>
```

Fig. 30.8 | Form for adding an entry to an address book. (Part 1 of 3.)

```
            <h:inputText id="lastNameInputText" required="true"
               requiredMessage="Please enter last name"
               value="#{addressBean.lastName}" maxlength="30"/>
            <h:message for="lastNameInputText" styleClass="error"/>
            <h:outputText value="Street:"/>
            <h:inputText id="streetInputText" required="true"
               requiredMessage="Please enter the street address"
               value="#{addressBean.street}" maxlength="150"/>
            <h:message for="streetInputText" styleClass="error"/>
            <h:outputText value="City:"/>
            <h:inputText id="cityInputText" required="true"
               requiredMessage="Please enter the city"
               value="#{addressBean.city}" maxlength="30"/>
            <h:message for="cityInputText" styleClass="error"/>
            <h:outputText value="State:"/>
            <h:inputText id="stateInputText" required="true"
               requiredMessage="Please enter state"
               value="#{addressBean.state}" maxlength="2"/>
            <h:message for="stateInputText" styleClass="error"/>
            <h:outputText value="Zipcode:"/>
            <h:inputText id="zipcodeInputText" required="true"
               requiredMessage="Please enter zipcode"
               value="#{addressBean.zipcode}" maxlength="5"/>
```

Fig. 30.8 | Form for adding an entry to an address book. (Part 2 of 3.)

```
                  <h:message for="zipcodeInputText" styleClass="error"/>
               </h:panelGrid>
               <h:commandButton value="Save Address"
                  action="#{addressBean.save}"/>
            </h:form>
            <h:outputLink value="index.xhtml">Return to Addresses</h:outputLink>
         </h:body>
      </html>
```

Fig. 30.8 | Form for adding an entry to an address book. (Part 3 of 3.)

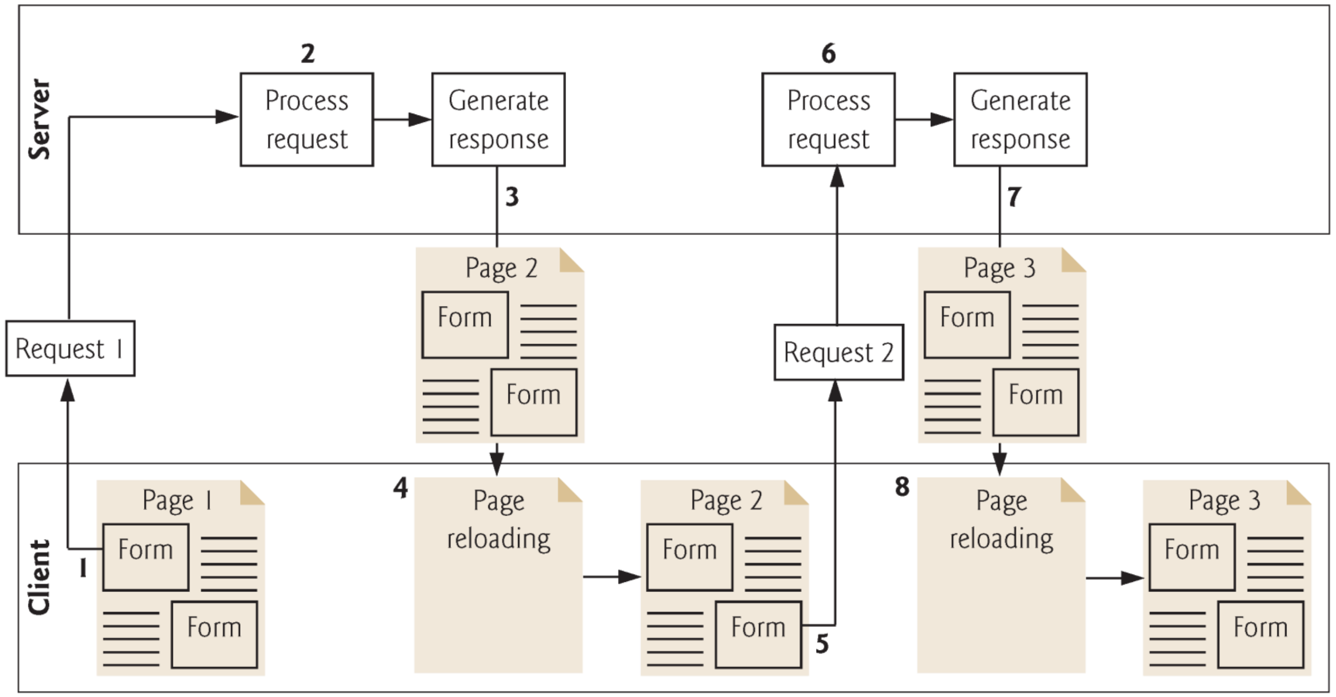


Fig. 30.9 | Classic web application reloading the page for every user interaction.

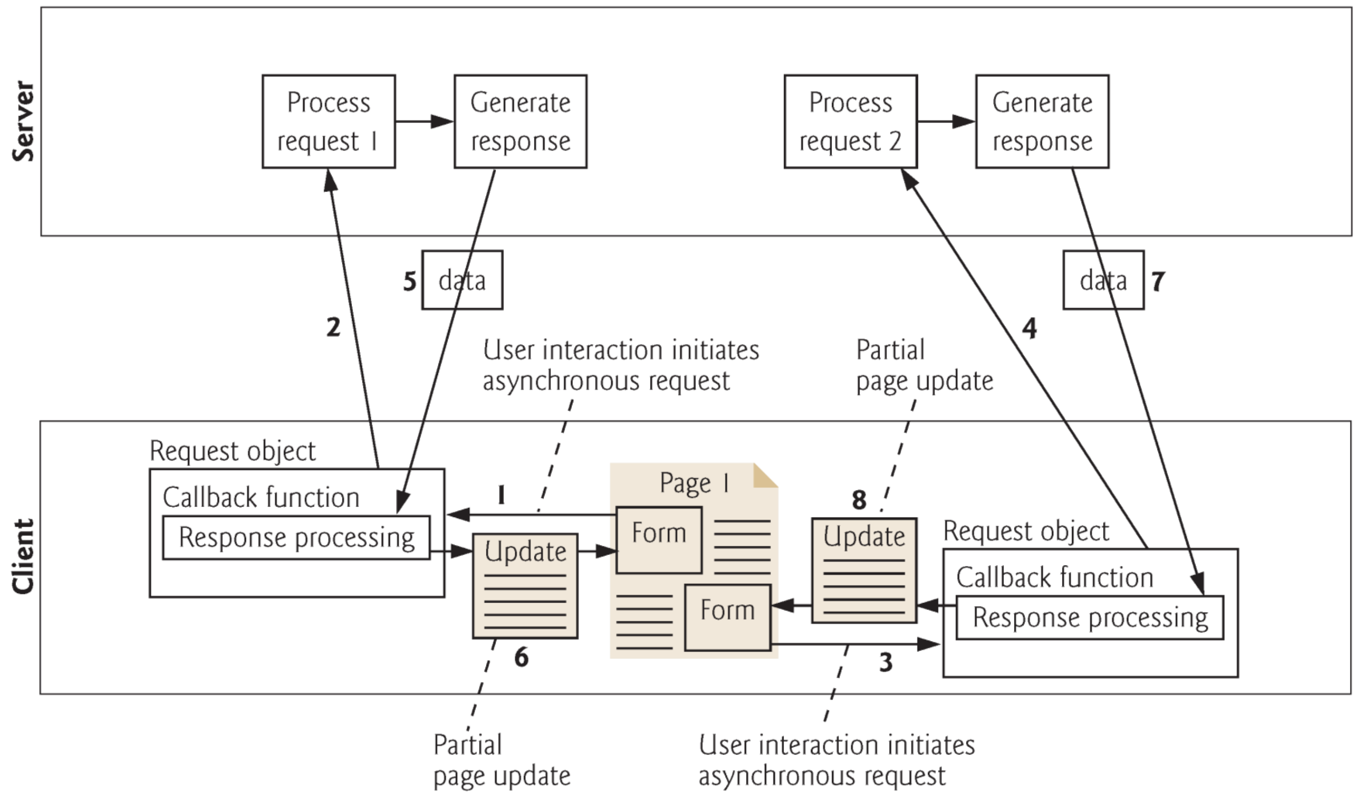
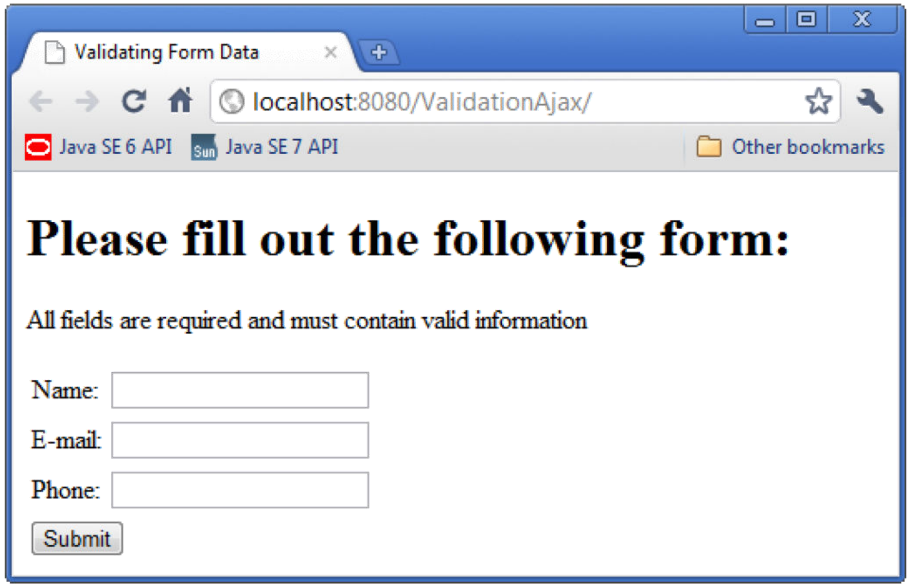


Fig. 30.10 | Ajax-enabled web application interacting with the server asynchronously.

a) Submitting the form before entering any information



Please fill out the following form:

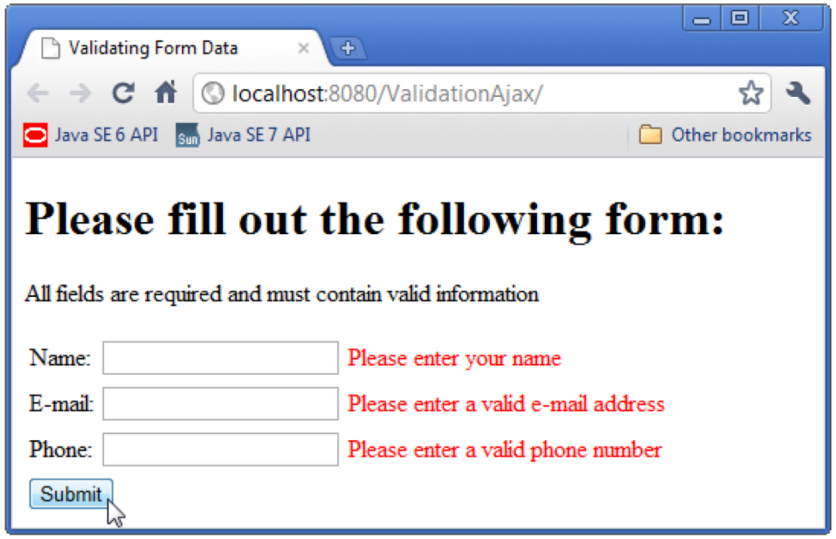
All fields are required and must contain valid information

Name:	
E-mail:	
Phone:	

Submit

Fig. 30.11 | JSP that demonstrates validation of user input. (Part 1 of 4.)

b) Error messages displayed after submitting the empty form



Please fill out the following form:

All fields are required and must contain valid information

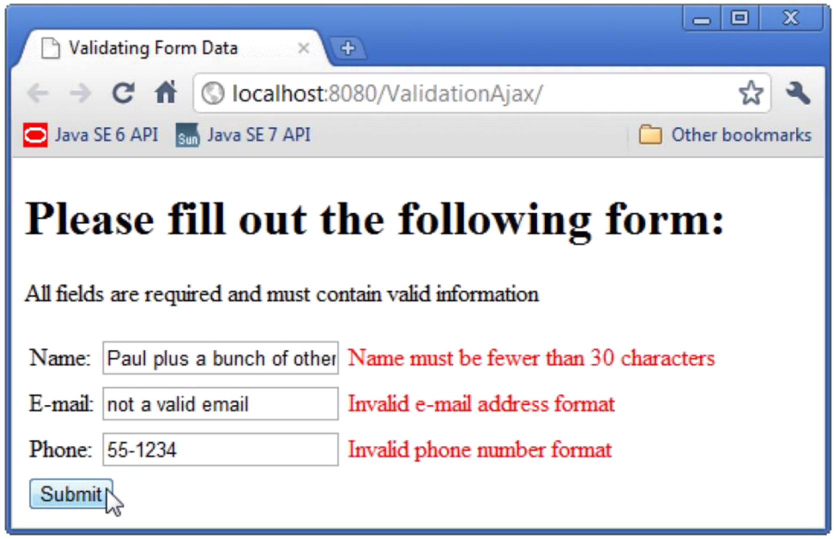
Name: Please enter your name

E-mail: Please enter a valid e-mail address

Phone: Please enter a valid phone number

Fig. 30.11 | JSP that demonstrates validation of user input. (Part 2 of 4.)

c) Error messages displayed after submitting invalid information



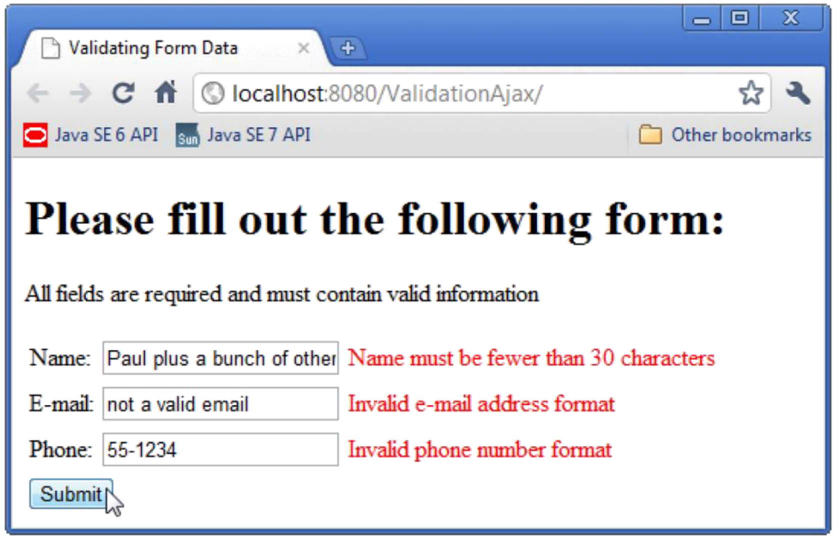
Please fill out the following form:

All fields are required and must contain valid information

Name:	Paul plus a bunch of other	Name must be fewer than 30 characters
E-mail:	not a valid email	Invalid e-mail address format
Phone:	55-1234	Invalid phone number format

Fig. 30.11 | JSP that demonstrates validation of user input. (Part 3 of 4.)

c) Error messages displayed after submitting invalid information



Please fill out the following form:

All fields are required and must contain valid information

Name:	Paul plus a bunch of other	Name must be fewer than 30 characters
E-mail:	not a valid email	Invalid e-mail address format
Phone:	55-1234	Invalid phone number format

Submit

Fig. 30.11 | JSP that demonstrates validation of user input. (Part 4 of 4.)

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

<!-- index.xhtml -->
<!-- Validating user input -->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:f="http://java.sun.com/jsf/core">
   <h:head>
      <title>Validating Form Data</title>
      <h:outputStylesheet name="style.css" library="css"/>
   </h:head>
   <h:body>
      <h:form>
         <h1>Please fill out the following form:</h1>
         <p>All fields are required and must contain valid information</p>
         <h:panelGrid columns="3">
            <h:outputText value="Name:"/>
            <h:inputText id="nameInputText" required="true"
               requiredMessage="Please enter your name"
               value="#{validationBean.name}"
               validatorMessage="Name must be fewer than 30 characters">
               <f:validateLength maximum="30" />
```

Fig. 30.12 | Ajax enabling the Validation app. (Part 1 of 3.)

```
               </h:inputText>
               <h:message id="nameMessage" for="nameInputText"
                  styleClass="error"/>
               <h:outputText value="E-mail:"/>
               <h:inputText id="emailInputText" required="true"
                  requiredMessage="Please enter a valid e-mail address"
                  value="#{validationBean.email}"
                  validatorMessage="Invalid e-mail address format">
                  <f:validateRegex pattern=
                     "\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*" />
               </h:inputText>
               <h:message id="emailMessage" for="emailInputText"
                  styleClass="error"/>
               <h:outputText value="Phone:"/>
               <h:inputText id="phoneInputText" required="true"
                  requiredMessage="Please enter a valid phone number"
                  value="#{validationBean.phone}"
                  validatorMessage="Invalid phone number format">
                  <f:validateRegex pattern=
                     "((\(\d{3}\) ?)|(\d{3}-))?\d{3}-\d{4}" />
               </h:inputText>
               <h:message id="phoneMessage" for="phoneInputText"
                  styleClass="error"/>
```

Fig. 30.12 | Ajax enabling the Validation app. (Part 2 of 3.)

```
            </h:panelGrid>
            <h:commandButton value="Submit">
               <f:ajax execute="nameInputText emailInputText phoneInputText"
                  render=
                  "nameMessage emailMessage phoneMessage resultOutputText"/>
            </h:commandButton>
            <h:outputText id="resultOutputText" escape="false"
               value="#{validationBean.response}"/>
         </h:form>
      </h:body>
   </html>
```

Fig. 30.12 | Ajax enabling the Validation app. (Part 3 of 3.)