# Java EE 6: Develop Web Applications with JSF

**Activity Guide**

D77738GC10

Edition 1.0

January 2014

D82091

## Authors

Paromita Dutta, Anjana Shenoy

## Technical Contributors and Reviewers

Feisal Ahmad, Syed Khadeer Ahmed, Joe Boulenouar, Edward Burns, Diganta Choudhury, John Clingan, Edgar Alberto Martinez Cruz, Tom Mc Ginn, Mathieu Heimer, Eduardo Moranchel Rosales

**This book was published using: Oracle Tutor**

# Table of Contents

# Practices for Lesson 1: Introduction to the Course

**Chapter 1**

# Practices for Lesson 1: Overview

## Overview

There are no practices for this Lesson.

# Practices for Lesson 2: Introducing JSF Technology

**Chapter 2**

# Practices for Lesson 2: Overview

## Practices Overview

In these practices, you will explore a simple JSF web application to gain a high-level understanding of the JSF framework and the structure of JSF applications. You will also deploy and run the application to become familiar with the run-time environment.

# Practice 2-1: Configuring NetBeans to Control WebLogic Server

## Overview

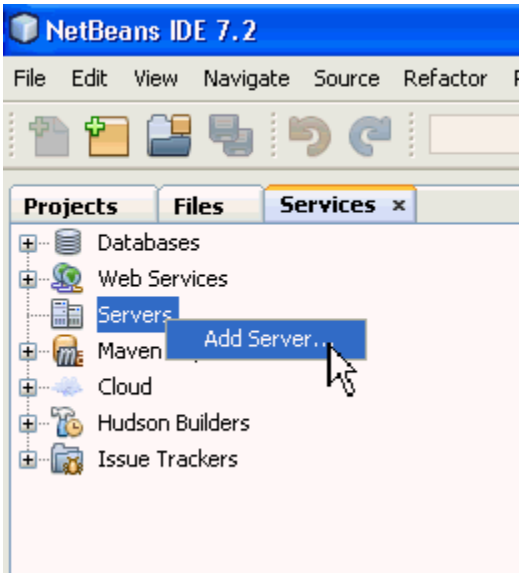
In this practice, you add a WebLogic Server instance to the NetBeans IDE. This will enable you to deploy Java EE applications to WebLogic Server from within NetBeans and also start or stop WebLogic Server from the Services tab in NetBeans.

## Assumptions

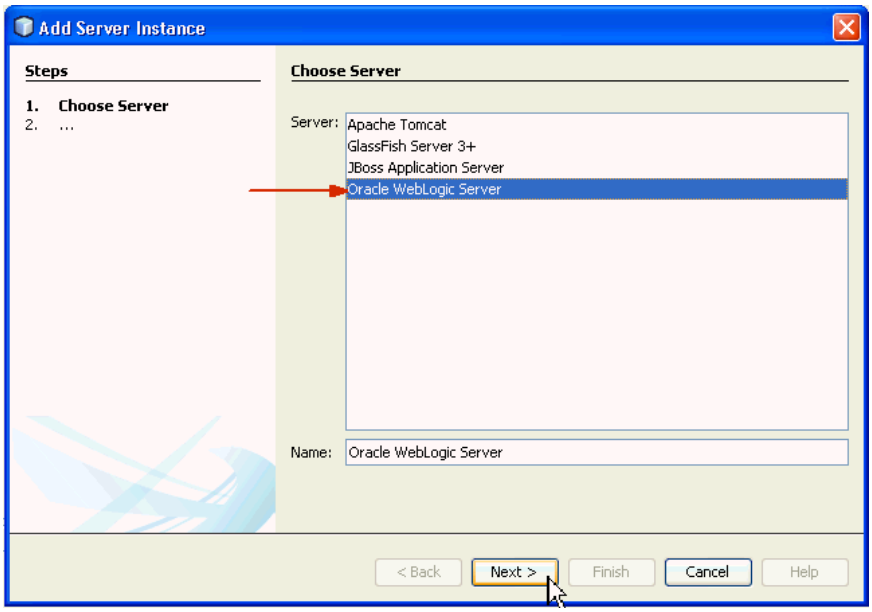
- JDK 7 is installed.
- NetBeans 7.2 EE is installed.
- Oracle WebLogic Server 12c (12.1.1) Zip Distribution is installed.

## Tasks

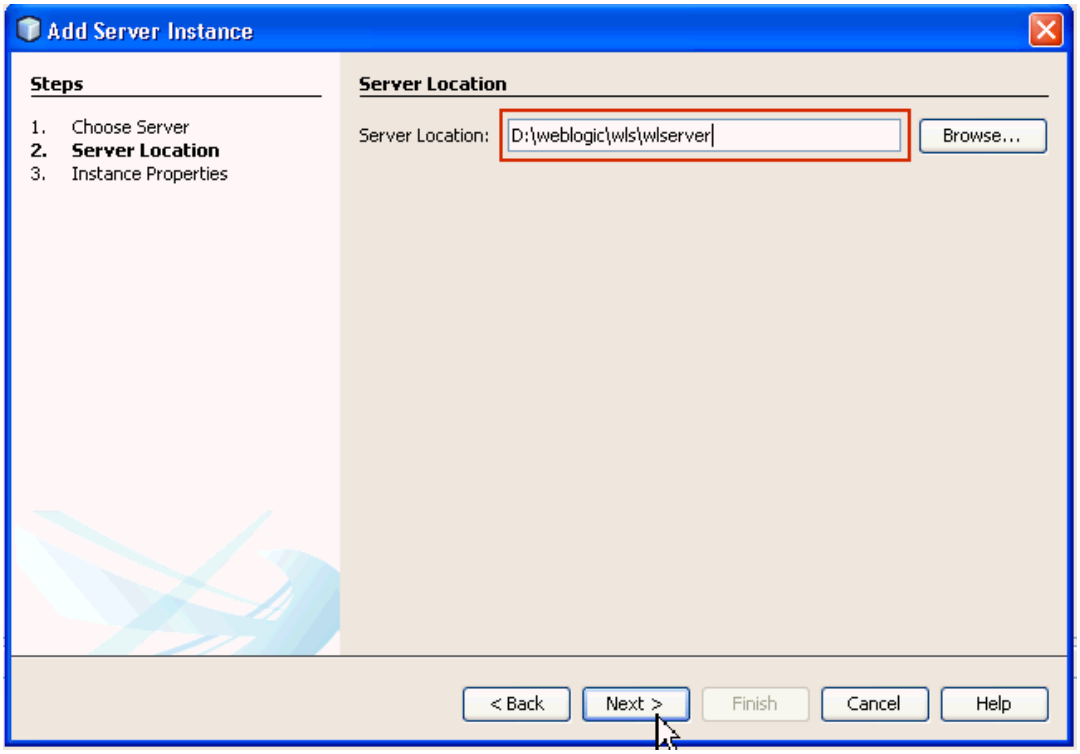
1. Start NetBeans.
2. Click the Services Tab and select the Servers node.
3. Right-click the Servers node and select Add Server.



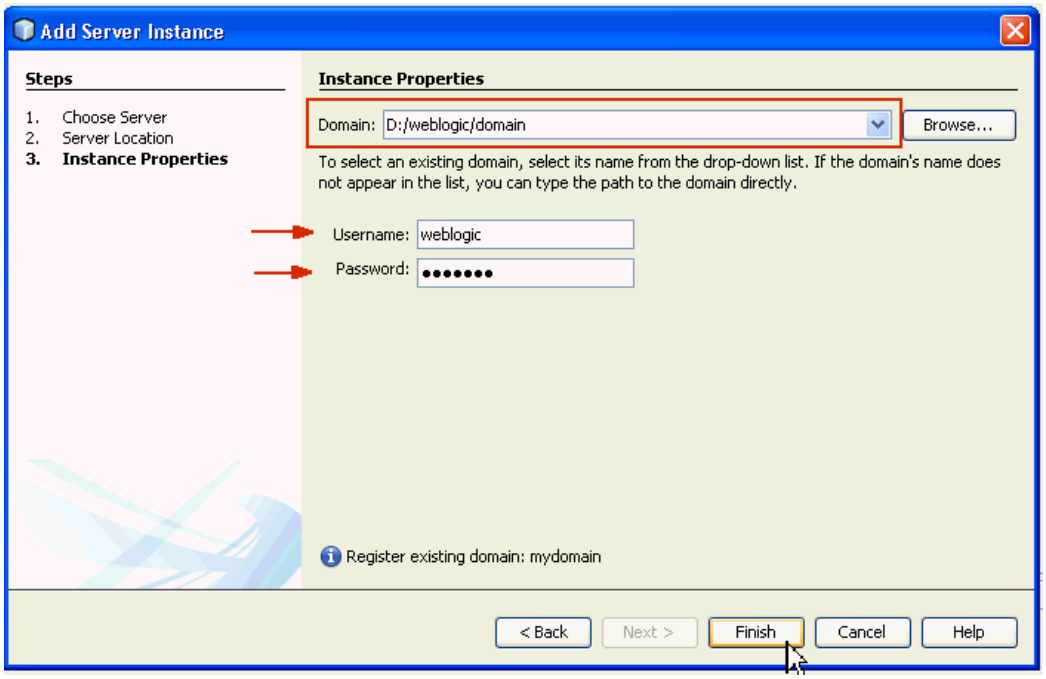
4. In the Add Server Instance window, select Oracle WebLogic Server and click Next.




5. Enter `D:\weblogic\wls\wlserver` for the server location and click Next.



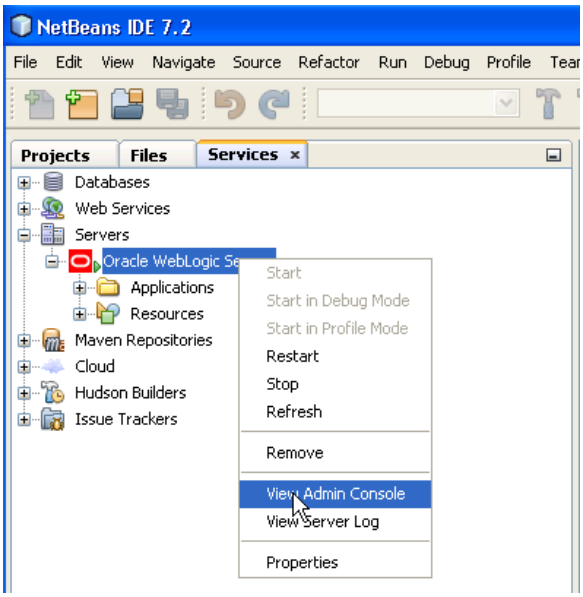
6. Verify that the domain location is `D:/weblogic/domain`, Enter `weblogic` and `welcome1` as the username and password, and click Finish.




7. Verify the configuration.
   - Expand the Servers node.
   - Right-click Oracle WebLogic Server and click Start.

   After WebLogic Server finishes loading, `<The server started in RUNNING mode.>` is shown in the output window.

8. Right-click Oracle WebLogic Server and select View Admin Console.




The login screen of the server opens in the browser.

9. Log in with the following credentials:
   - username: `weblogic`
   - password: `welcome1`

   You will see Home Page of the Admin Console in your browser.

# Practice 2-2: Exploring JSF Web Applications

## Overview

In this practice, you explore a simple JSF web application and gain a high-level understanding of the JSF framework and the structure of JSF applications. You also deploy and run the application to become familiar with the Integrated Development Environment (IDE).

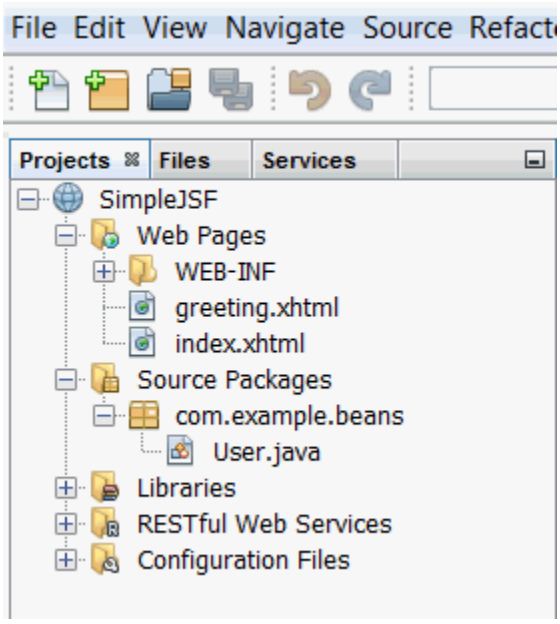
## Assumptions

NetBeans is running.

WebLogic Server is started.

## Tasks

1. Open the `SimpleJSF` project in NetBeans
   a. In NetBeans, from the File menu, select Open Project.
   b. Browse to the path-
      `D:\labs\02_IntroJSF\practices\practice2` folder.
   c. Select the project and click Open Project.
2. Look at the files in the project.
   a. In the Projects tab, expand the `SimpleJSF` project.
   b. Expand the Web Pages folder. The structure of the Web Pages directory looks similar to the following:




   c. In the NetBeans IDE, a web application is organized as follows:
      1) The **Web Pages** node is the root for all the JSF pages, and it also contains the WEB-INF node that contains the deployment descriptors for the application. Notice that these deployment descriptors are also referenced in the Configuration Files node.

2) The **Source Packages** node is the root for the Java source files and the resource bundles.
3) The **Libraries** node is the root of the run-time library for the application.
4) **Configuration Files** is the root of the deployment descriptors and additional configuration information, such as the manifest files used in creating the Java Archive (JAR) files.

3. Open the `index.xhtml` page.

   The XHTML-based Facelets page, `index.xhtml`, is the welcome page of the application. Its contents are listed as follows:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!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>Simple JSF 2.0 Application</title>
  </h:head>
  <h:body>
     <h2>Simple JSF 2.0 Application</h2>
     <h:form>
        Please enter your name: <h:inputText value="#{user.name}" />
     <br/>
     <h:commandButton value="Submit" action="greeting"/>
     </h:form>
  </h:body>
</html>
```

4. Open the `greeting.xhtml` page.

   The `greeting.xhtml` page is navigated to after the user clicks the command button labeled Submit:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!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>Simple JSF 2.0 Application</title>
  </h:head>
    <h:body>
      <h2>Simple JSF 2.0 Application</h2>
      <p>
       Welcome, <b>#{user.name}</b>. Hope you enjoy JSF 2.0.
      </p>
   </h:body>
</html>
```

   a. Note the `#{user.name}` syntax. This returns a string `name` from the managed bean named “user.” JSF looks for any bean either named “user” through injection or by virtue of its class name.

5. Examine `User.java`.

   a. In the Projects tab, select `SimpleJSF` project.

   b. Expand the `Source Packages > com.example.beans` and select `User.java`.

   c. Double-click `User.java` to open it in the code editor.

   d. Observe the following:

      1) This class is a backing or managed bean for this application. It contains the String name property that the index page writes to with the `inputText` component, and reads from on the greeting page.

2) Note that any managed bean property will have a `set<property name>` method to write to the property and a `get<property name>` method to read the current value of the property.

```
@Named
@RequestScoped
public class User {
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String string) {
        this.name = string;
    }
}
```

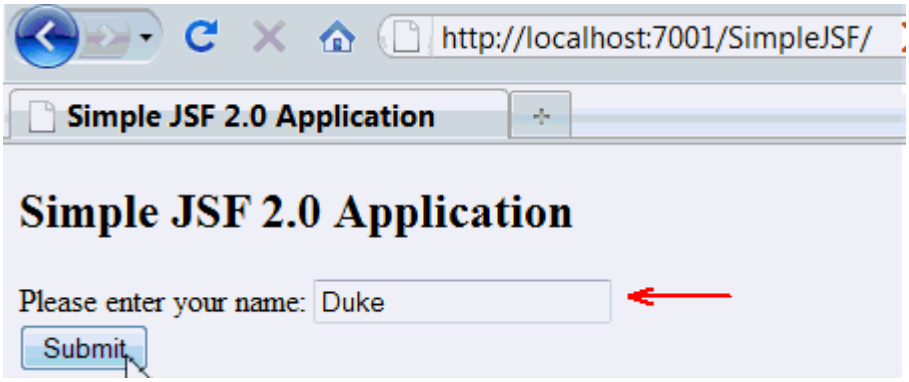
6. Build, Deploy and Run `SimpleJSF` Web Application.
   a. Select **`SimpleJSF`** in the Projects tab.
   b. Right-click the project and select **Clean and Build.**
   c. Right-click the project and select **Deploy**.
   d. To run the application, right-click the project and select **Run**.
   e. After a while, your default Web browser is started and the following page is shown.


**Simple JSF 2.0 Application**

Please enter your name:

Submit

1) Enter a word in the input box and click **Submit**.



2) You should see a response page as shown below

# Practice 2-3: Creating the HelloJSF Web Application

## Overview

In this practice, you create a simple JSF web application to enhance your understanding of the JSF web application structure. You will also become more familiar with the development environment.

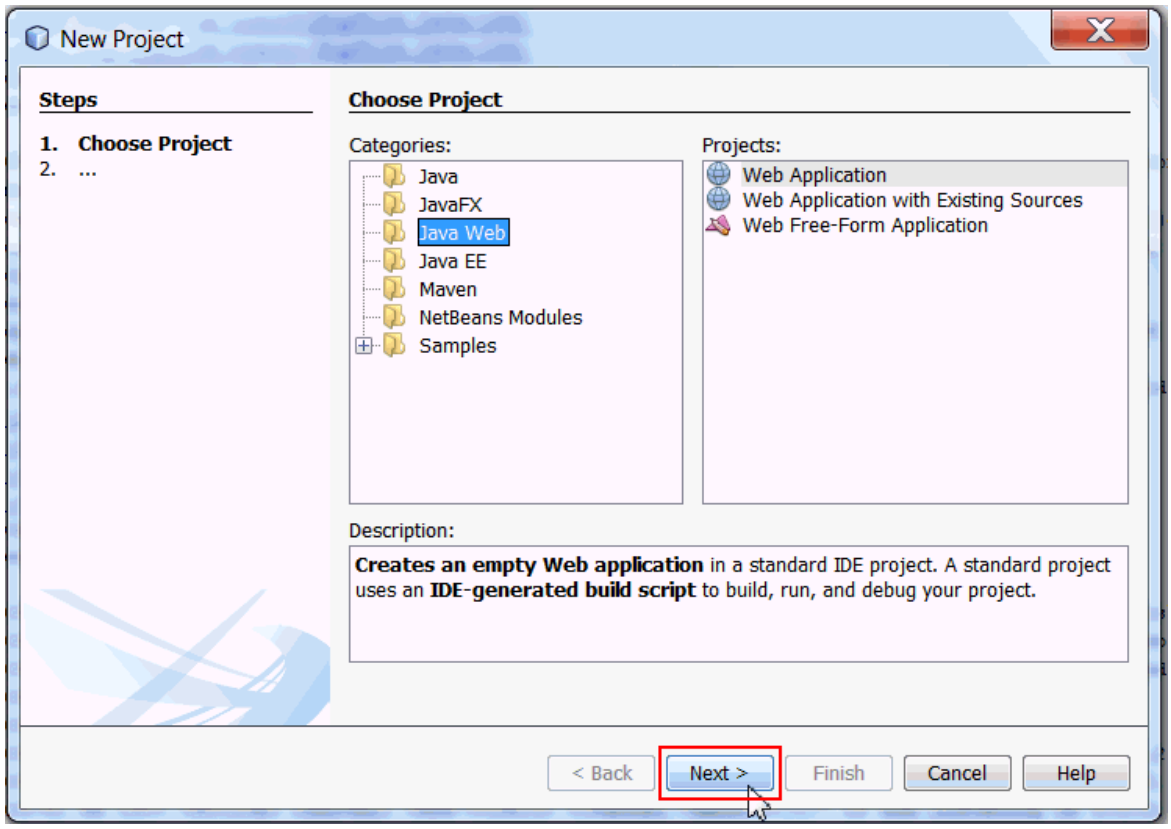
## Assumptions

NetBeans is running.

WebLogic Server is started.

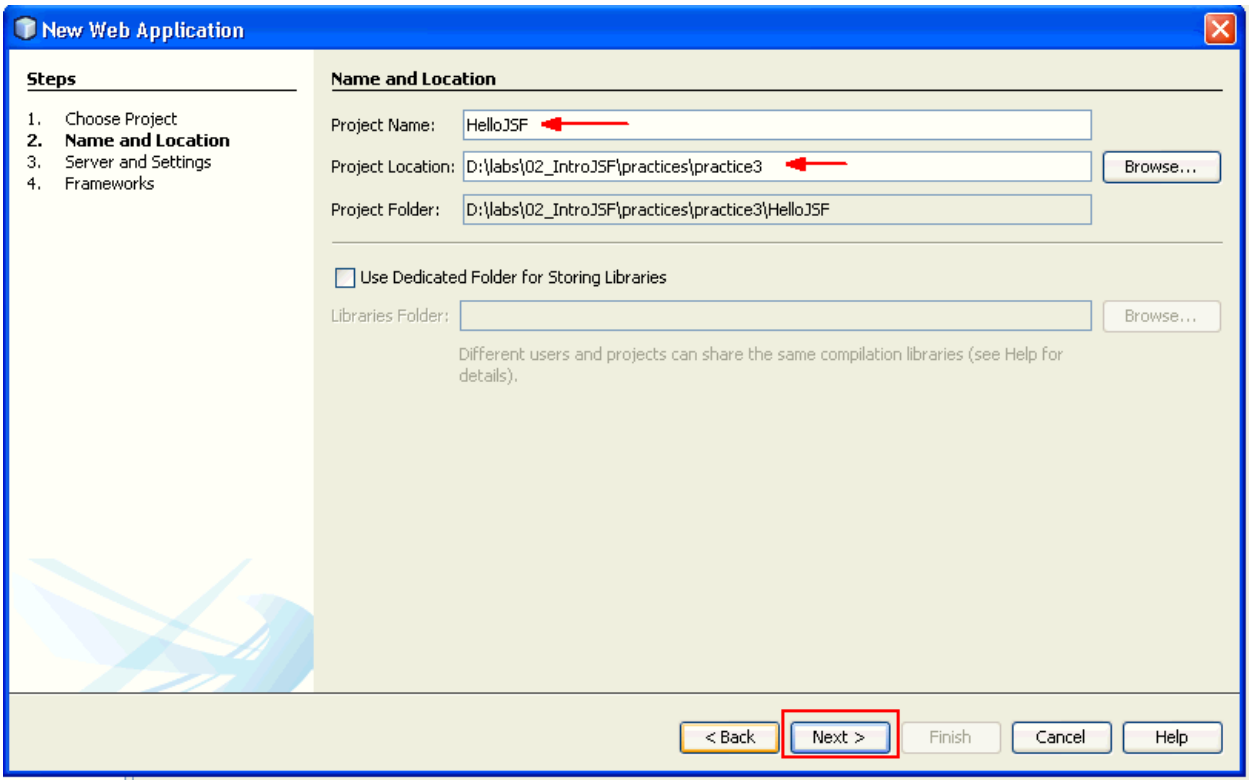
## Tasks

1. Create a new project in NetBeans.
   a. Click **File**, and then select **New Project**.
   b. In the New Project Wizard, select the **Java Web** category.
   c. Under Projects, select **Web Application** and click **Next**.




   d. Enter the project name as `HelloJSF`
   e. Specify the location for the project as `D:\labs\02_IntroJSF\practices\practice3`

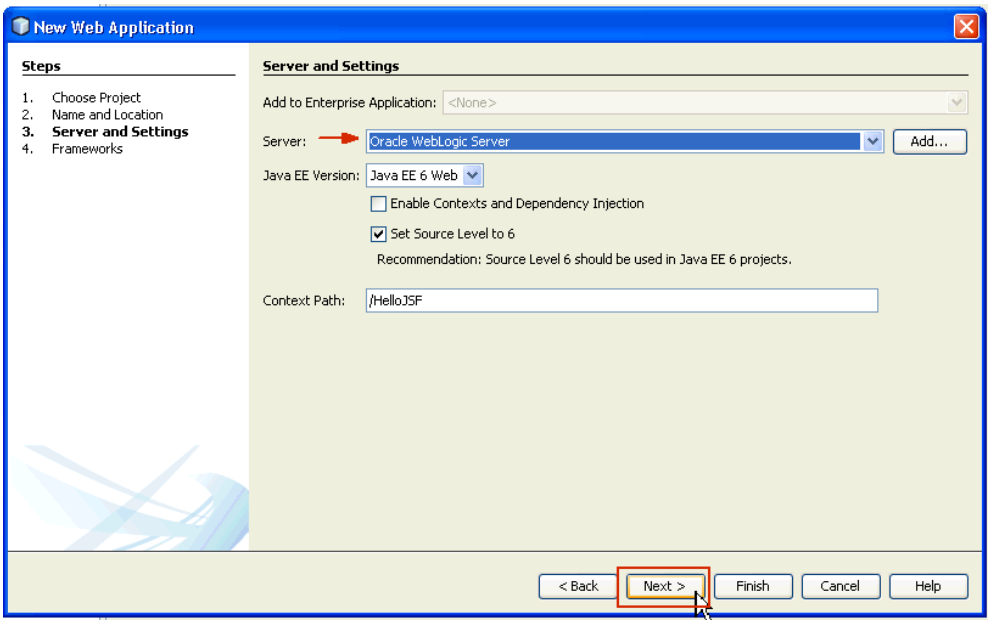
f. Click **Next**.




g. In the Server and Settings Window,

1) Ensure the Server is set to Oracle WebLogic Server.

2) Click Next.

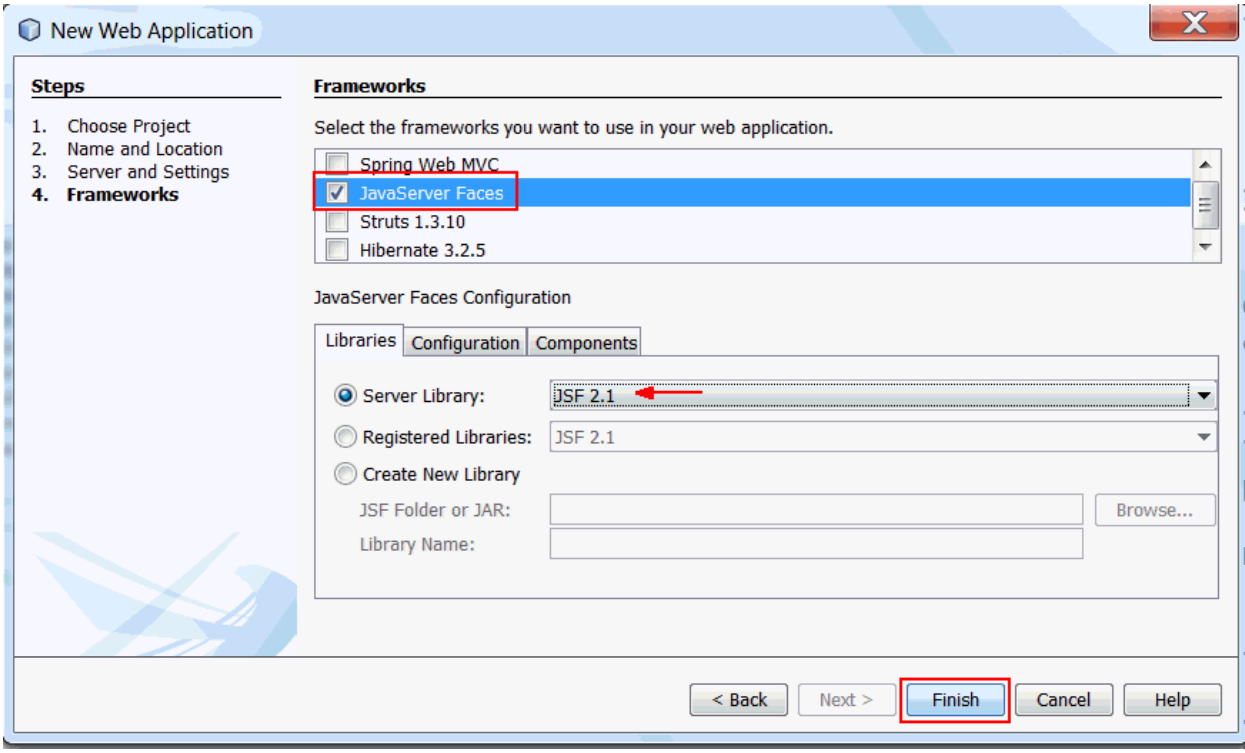


h. In the Frameworks window, perform the following steps:

1) In the Frameworks panel, select **JavaServer Faces**.

2) In the JavaServer Faces Configuration panel, select **JSF 2.1** for the Server Library option from the drop down list.

3) Click **Finish**.




The IDE generates a new project and adds the JSF-specific configuration files to the project as necessary.

2. Edit the `index.xhtml` page as follows:

a. Double-click `index.xhtml` to open in the editor.

b. Change the title element to a different value,

```
<title>The JSF version of HelloWorld</title>
```

c. Edit the `<h:body>` element by adding a form element with the following content:

```
<h:form>
Please enter a word: <h:inputText value="#{word}"/>
<h:commandButton value="Go" action="response"/>
</h:form>
```

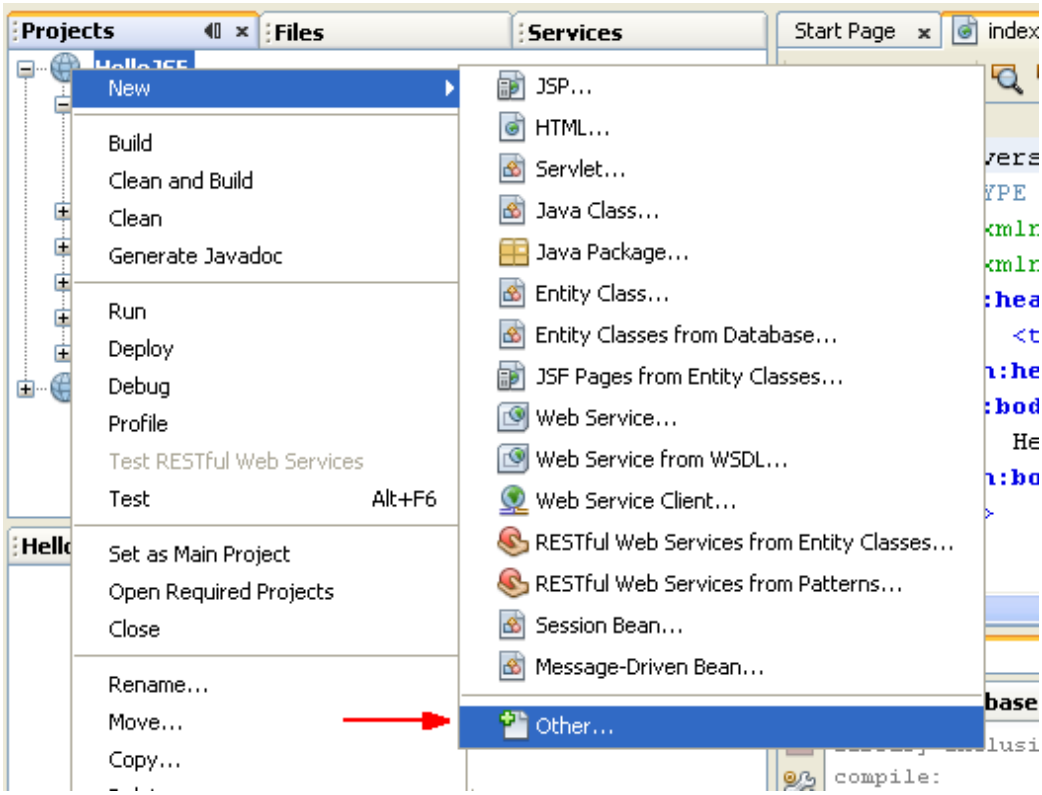
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>The JSF version of HelloWorld</title>
    </h:head>
    <h:body>
        <h:form>
            Please enter a word: <h:inputText value="#{word}"/>
            <h:commandButton value="Go" action="response"/>
        </h:form>
    </h:body>
</html>
```

3. Save the page.

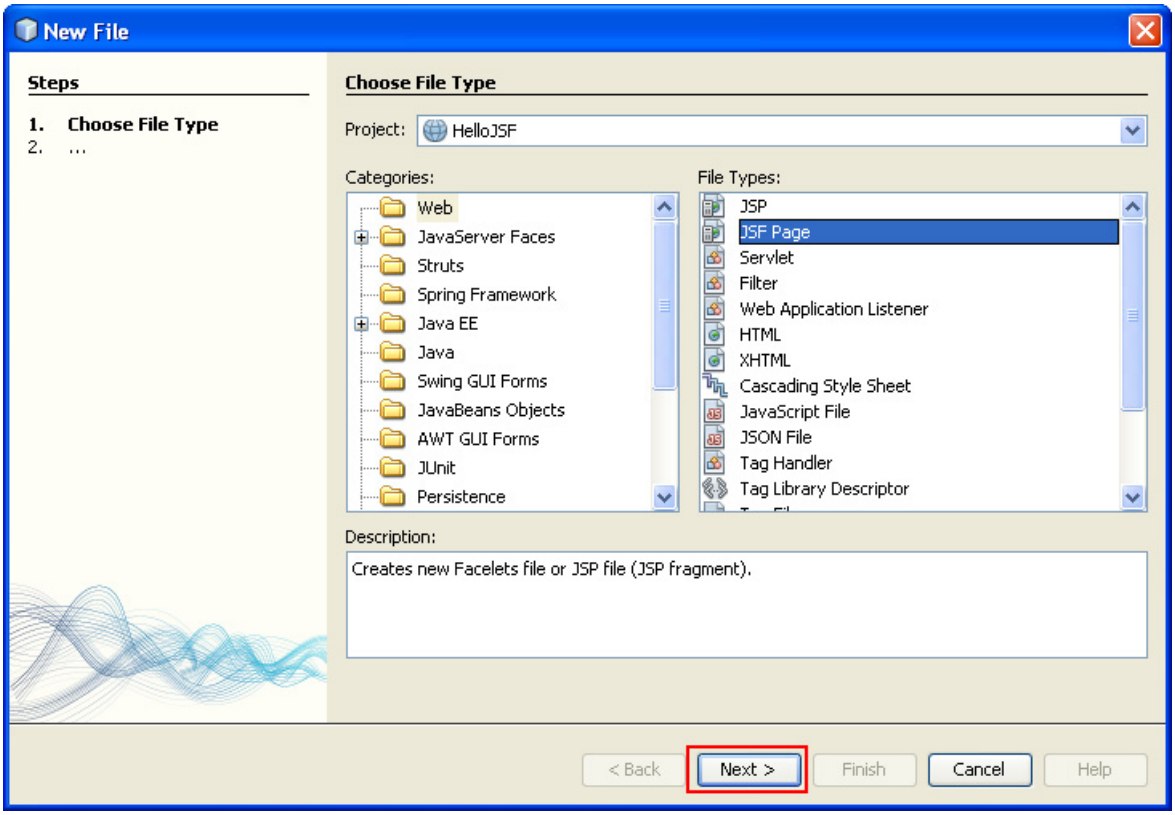
   The Expression Language (EL) value expression `#{word}` defines an attribute named "word" in the default request scope.

4. Create a new JSF page by performing the following:

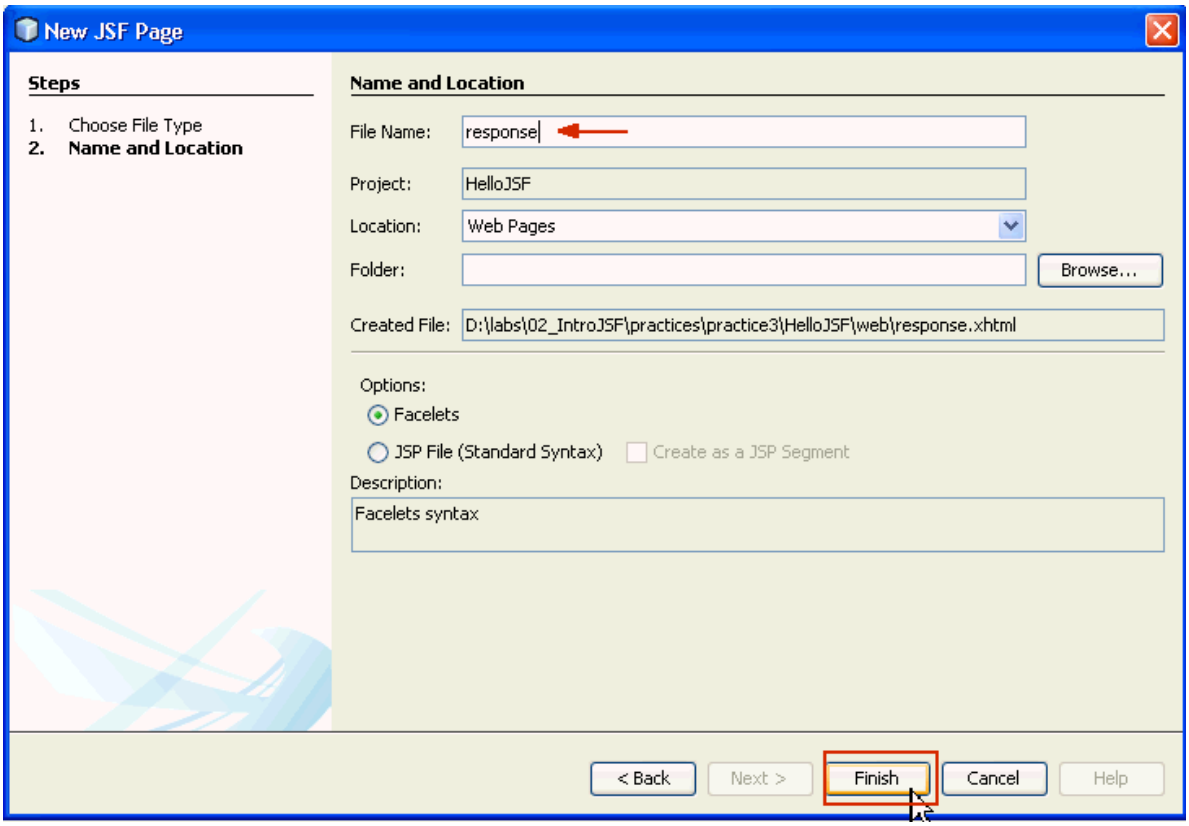
   a. In the Projects tab, right-click **HelloJSF** project and select **New > Other**:



b. When the New File dialog box appears, select **Web** as the Category and **JSF Page** as the file type, and then click **Next**.



c. In the New JSF File dialog box, enter the name of the new JSF page, **`response`** and click **Finish**.




5. Open the `response.xhtml` page and make the following changes:

   a. Change the title element to a different value, such as:

```
<title>Response of the JSF Hello World Application</title>
```

   b. Edit the `<h:body>` element of the page by adding a form element with the following content:

```
You entered: <b>#{word}</b>
```

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "ht
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>Response of the JSF Hello World Application</title>
    </h:head>
    <h:body>
        You entered: <b>#{word}</b>
    </h:body>
</html>
```

c. Save the page.

6. Build, Deploy and Run `HelloJSF` Web Application.

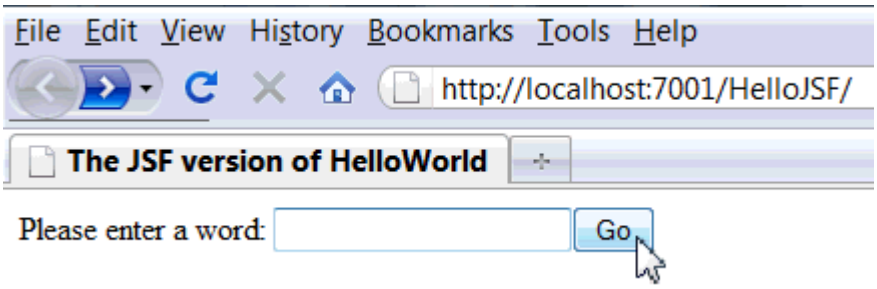
   a. Select `HelloJSF` in the Projects tab.

   b. Right-click the project and select **Build.**

   c. Right-click the project and select **Deploy**.

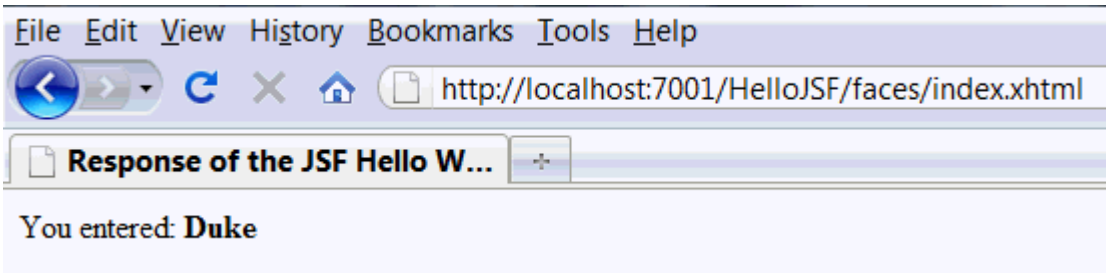
   d. To run the application, right-click the project and select **Run**.

   e. After a while, your default Web browser is started and the following page is shown.




   f. Enter a word in the input box and click **Go**.

   g. You should see a response page that contains the word you entered on the previous page.



# Practices for Lesson 3: Creating JSF Pages Using Facelets

**Chapter 3**

# Practices for Lesson 3: Overview

## Practices Overview

In these practices, you will create JSF pages by using Facelets, including using the core and HTML Render Kit tag libraries, and use application messages.

**Note:** From this point onward in the practices, you will be building the DVDLibrary application incrementally in a projects folder. If you need to go back to a previous exercise, or if you miss an exercise, you can copy the solution at the appropriate step from the solutions directory.

# Practice 3-1: Creating the `DVDLibrary` Application

## Overview

In this practice, you work with a web application called `DVDLibrary`. This web application allows users to log on and manage the DVDs in a DVD collection. The `DVDLibrary` application is incrementally extended in the following practicess to implement more capabilities.

## Assumptions

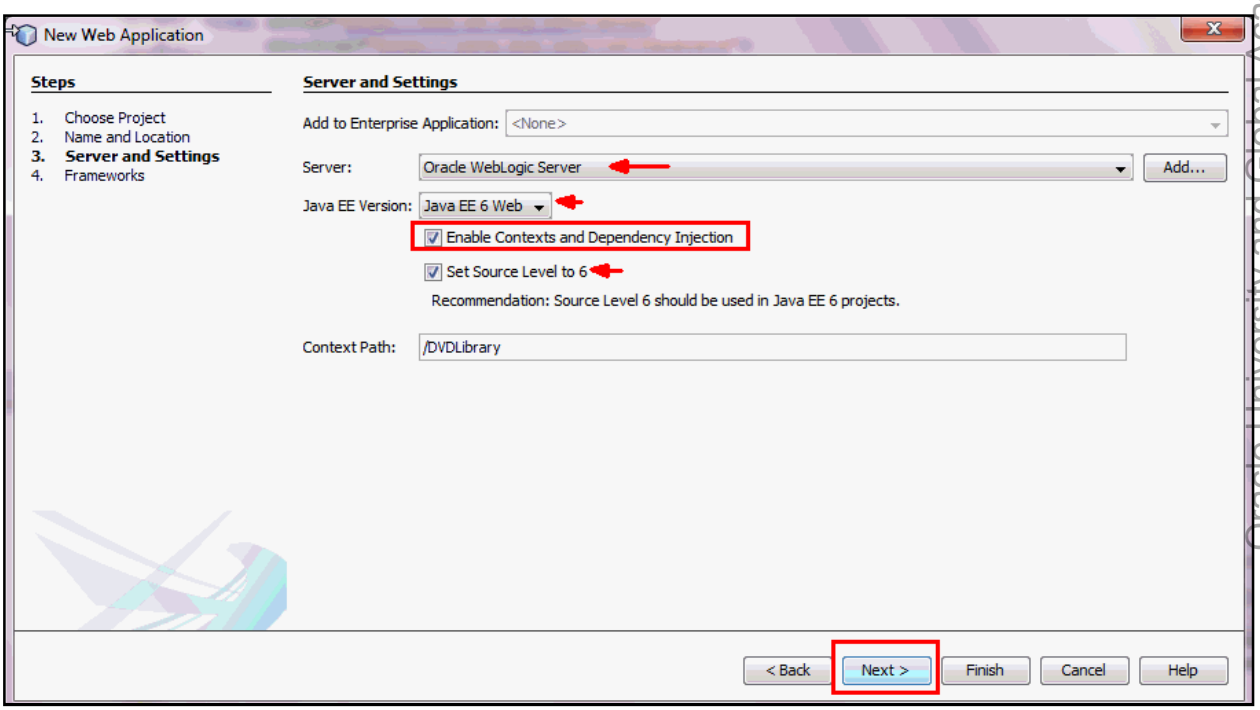
NetBeans is running.

## Tasks

1. Create the `DVDLibrary` web application with the JSF framework by using the following project-specific information:
   a. In NetBeans, select File -> New Project from the Menu bar.
   b. In the New Projects dialog box, select **Java Web** in the Categories section and **Web Application** in the Projects section. Click Next.
   c. Enter the following information in the Name and Location section and click Next.

| | |
|---|---|
| Project Name | `DVDLibrary` |
| Project Location | `D:\labs\03_Facelets\practices` |

   d. In the Server and Settings section, select the details as shown in the following screenshot and click Next.




   e. In the Frameworks section, select **Java Server Faces** and click Finish.
   f. Verify if the `DVDLibrary` project is visible in the Projects window and `index.xhtml` opens in the code editor window.

2. Create a new JSF page named `login.xhtml`. Complete the following steps:
   a. Right-click `DVDLibrary` in the Projects window and select New > Other.
   b. In the Choose File Type section, select Categories >Web and File Types >JSF Page. Click Next.
   c. In the Name and Location section, enter `login` as the File Name and click Finish.
3. Build the `DVDLibrary` application.
4. Deploy the `DVDLibrary` application.
5. Access the following URLs of the application to verify that all the pages are created:

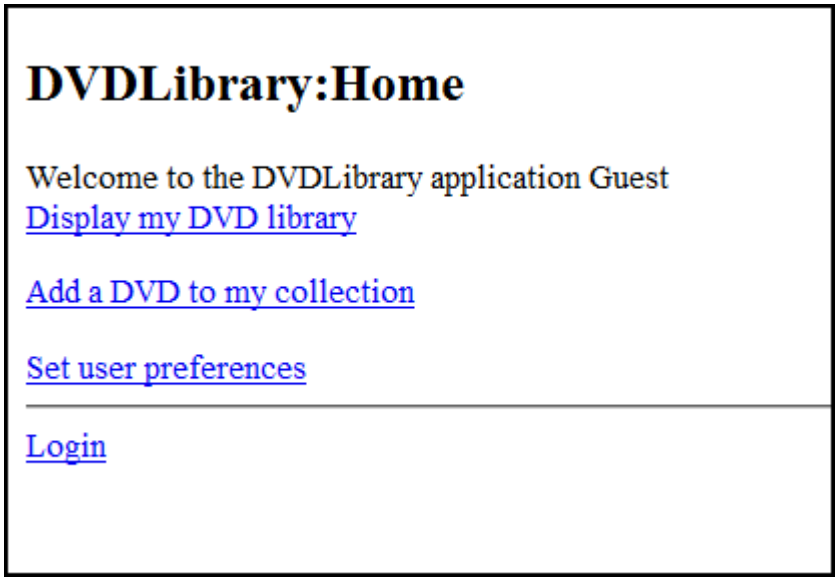
```
http://localhost:7001/DVDLibrary
http://localhost:7001/DVDLibrary/faces/index.xhtml
http://localhost:7001/DVDLibrary/faces/login.xhtml
```

## Practice 3-2: Designing the Home Page `index.xhtml`

### Overview

In this practice, you design the JSF pages created in the previous practice by using the HTML Render Kit tag libraries.

The home page will look as shown in the following image:




### Assumptions

NetBeans is running and you have completed Practice 1.

### Tasks

1. In the `DVDLibrary` project, open the JSF page `index.xhtml`.
2. Change the value of the title element to the following:

```
<title>DVD Library Application</title>
```

3. Add a level-2 heading element to the body of the page, for example:

```
<h2>DVDLibrary:Home</h2>
```

4. Add a short welcome message to the body of the page, for example:

```
Welcome to the DVD Library Application!
```

5. Add a `form` element to the body of the page.

6. Within the form, add four `commandLink` elements and separate the first three elements from the last with a line break, for example:

```
<h:form>
    <h:commandLink>Display my DVD library</h:commandLink>
    <p/>
    <h:commandLink>Add a DVD to my collection</h:commandLink>
    <p/>
    <h:commandLink>Set user preferences</h:commandLink>
    <hr/>
    <h:commandLink>Login</h:commandLink>
</h:form>
```

7. Save the `index.xhtml` page.
8. Clean and build the `DVDLibrary` project. Run the `DVDLibrary` application, and then access the URL for the home page:

   `http://localhost:7001/DVDLibrary`

## Practice 3-3: Designing the Login Page `login.xhtml`

### Overview

In this practice, you will design the login page of the `DVDLibrary` application to add a login form.

The login page will look as shown in the following image:


### Assumptions

You have completed the previous two practices.

### Tasks

1. In the `DVDLibrary` project, open the JSF page `login.xhtml`.
2. Change the value of the title element to the following:

```
<title>The DVDLibrary Application</title>
```

3. Add a level-2 heading element to the body as follows:

```
<h2>DVDLibrary:Login</h2>
```

4. Add a `<h:form>` element to the body of the page.
5. Add a `<h:panelGrid>` tag to the form element, and set the column attribute to 2.
6. Within the panel grid of the form, add the following components sequentially:

```
User Name: <h:inputText/>
Password: <h:inputSecret/>
<h:commandButton value="Login"/>
```

7. Save the `login.xhtml` page.
8. Clean and build the `DVDLibrary` project. Run the `DVDLibrary` application.
9. Access the URL for the home page:

   `http://localhost:7001/DVDLibrary/faces/login.xhtml`.

# Practices for Lesson 4: Creating the DVDLibrary CDI Beans

**Chapter 4**

# Practices for Lesson 4: Overview

## Practices Overview

In these practices, you will create a managed bean for the DVDLibrary project, use the classes provided to connect to a database, and implement functionality to add a new record to the database.

# Practice 4-1: Creating the `LoginBean` Context Dependency Injection (CDI) Bean

## Overview

In this practice, you create a `LoginBean` CDI bean object to represent the user of the application.

## Assumptions

NetBeans is running and you should have completed practices of the previous lesson.

## Tasks

1. Create a JSF CDI Bean:
   a. Right-click the DVDLibrary project and select **New**.
   b. Select **Java Class** and click **Next**.
   c. Enter the following information on the New Java class dialog box:

| | |
|---|---|
| Class Name | `LoginBean` |
| Location | `Source Packages (default)` |
| Package | `com.example.beans` |

After the managed bean is created, you should see the following Java class, `LoginBean`, created in the `com.example.beans` source package:

```
package com.example.beans;

public class LoginBean {

}
```

2. Edit the `LoginBean` class as follows:
   a. Edit the class to implement Serializable and add the appropriate import statement:

```
import java.io.Serializable;
public class LoginBean implements Serializable {
```

   b. Add two annotations to make the Bean a CDI Named Bean and Session Scoped above the class declaration:

```
@Named("login")
@SessionScoped
```

   c. These will require additional import statements:

```
import javax.inject.Named;
import javax.enterprise.context.SessionScoped;
```

d. Add the following two properties to the class:

```
private String username;
private String password;
```

e. Create a public getter and setter for each property:

**Note:** NetBeans has a simple method for adding getter and setter methods. Right-click the line where you want the methods to appear and select **Insert Code**. Select **Getter and Setter** from the **Generate** menu and select the fields that you want NetBeans to generate methods for, in this case, `username` and `password`.

```
    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }
```

f. Save the `LoginBean` class.

## Practice 4-2: Updating the Home Page to Use `LoginBean`

### Overview

In this practice, you update the `index.xhtml` JSF page created to interact with `LoginBean`.

**Note:** The `index.xhtml` JSF page was created in Practice 2, "Creating the DVDLibrary Application."

### Assumptions

NetBeans is running and you should have completed the previous practice.

### Tasks

1. Open the `index.xhtml` page.
2. Instead of the current static welcome message, return the value of the username property from `DVDLibraryBean` by replacing the "Welcome to the DVD Library Application" with the following:

```
Welcome to the DVDLibrary application
<h:outputFormat value="#{empty login.username ? 'Guest' :
login.username}"/>
!
```

   This code uses the conditional operator of the unified EL to use the static text `Guest` if the `username` property of the `dvd` managed bean is empty, and use the actual value of the `username` property otherwise.

3. Save the `index.xhtml` page and run the `DVDLibrary` application to test your changes.

# Practice 4-3: Updating the Login Page to Use LoginBean

## Overview

In this practice, you update the `login.xhtml` page created in Practice 2, "Creating the DVDLibrary Application," to interact with `LoginBean`.

## Assumptions

NetBeans is running and you should have completed the previous practices.

## Tasks

1. Open the `login.xhtml` page.
2. Locate the `inputText` element and add the value attribute to bind to the username property of the `dvd` managed bean.

   ```
   <h:inputText value="#{login.username}"/>
   ```

3. Locate the `inputSecret` element and add the value attribute to bind to the password property of the `dvd` managed bean:

   ```
   <h:inputSecret value="#{login.password}"/>
   ```

4. Update the last `commandButton` element in the form by adding an action attribute with a value of index:

   ```
   <h:commandButton value="Login" action="index"/>
   ```

   **Note:** In this example, the action attribute triggers an implicit navigation rule that results in the home page being rendered as the response. You will work with page navigation in the practice titled "Working with Navigation."

5. Save `login.xhtml`.
6. Clean and Build the `DVDLibrary` application.
7. Deploy the `DVDLibrary` application.
8. Access the URL for the home page:

   ```
   http://localhost:7001/DVDLibrary
   ```

9. The home page should display the following welcome message, indicating that the `username` property of the `dvd` managed bean has not yet been set:

   ```
   Welcome to the DVDLibrary application Guest
   ```

10. Access the `login.xhtml` page by using the following URL:

    ```
    http://localhost:7001/DVDLibrary/faces/login.xhtml
    ```

11. Enter an arbitrary username and password combination, and click **Login**.

    The browser should render the home page as the response. This time, because the username and password properties of the bean have been set, the welcome message should include the username that you entered, for example:

    ```
    Welcome to the DVDLibrary application Tom
    ```

# Practice 4-4: Setting Up the Database and JPA Files

## Overview

In this practice, you configure a database to serve as the DVD Library data store, and you use the `DVDLibraryBean` managed bean to interact with the database. When a DVD is added to the library, a new DVD item record is persisted in the database.

## Assumptions

NetBeans is running and you should have completed the previous practice.

## Tasks

1. Click the **Services** tab.
2. Stop the JavaDB database server if it is running.
   a. Right-click the **Java DB** icon.
   b. Select **Stop Server**.
3. Open the Properties menu for Java DB.
   a. Right-click the **Java DB** icon and select **Properties**.
   b. Make sure that the Java DB Installation directory is: `D:\Program Files\Java\jdk1.7.0_07\db`. This is the location of the JavaDB executable.
   c. Set the database path to `D:\Program Files\Java\jdk1.7.0_07\db \database`. This will be the location of the `dvdlibrary` database. Create a new `database` folder if required.
   d. Start the Java DB server.
      1) Right-click the **Java DB** icon.
      2) Select Start Server.

4. Create the database. Create tables and insert data.
   **a.** Right-click the **Java DB** icon and select **Create DataBase.**
   b. Enter the following values and click OK:

| | |
|---|---|
| Name | `dvdlibrary` |
| User name | `oracle` |
| Password | `oracle` |

   c. Verify if a connection to the `dvdlibrary` database is created.
   d. Right-click the `jdbc:derby://localhost:1527/dvdlibrary [oracle on ORACLE]` connection and select **Connect.**
   e. Select **File >Open File** from the Menu bar and open the `dvdlibrary.sql` file located at `D:\labs\resources\jpa` folder. The file opens in the SQL command window.

f. Select the `jdbc:derby://localhost:1527/dvdlibrary [oracle on ORACLE]` connection.

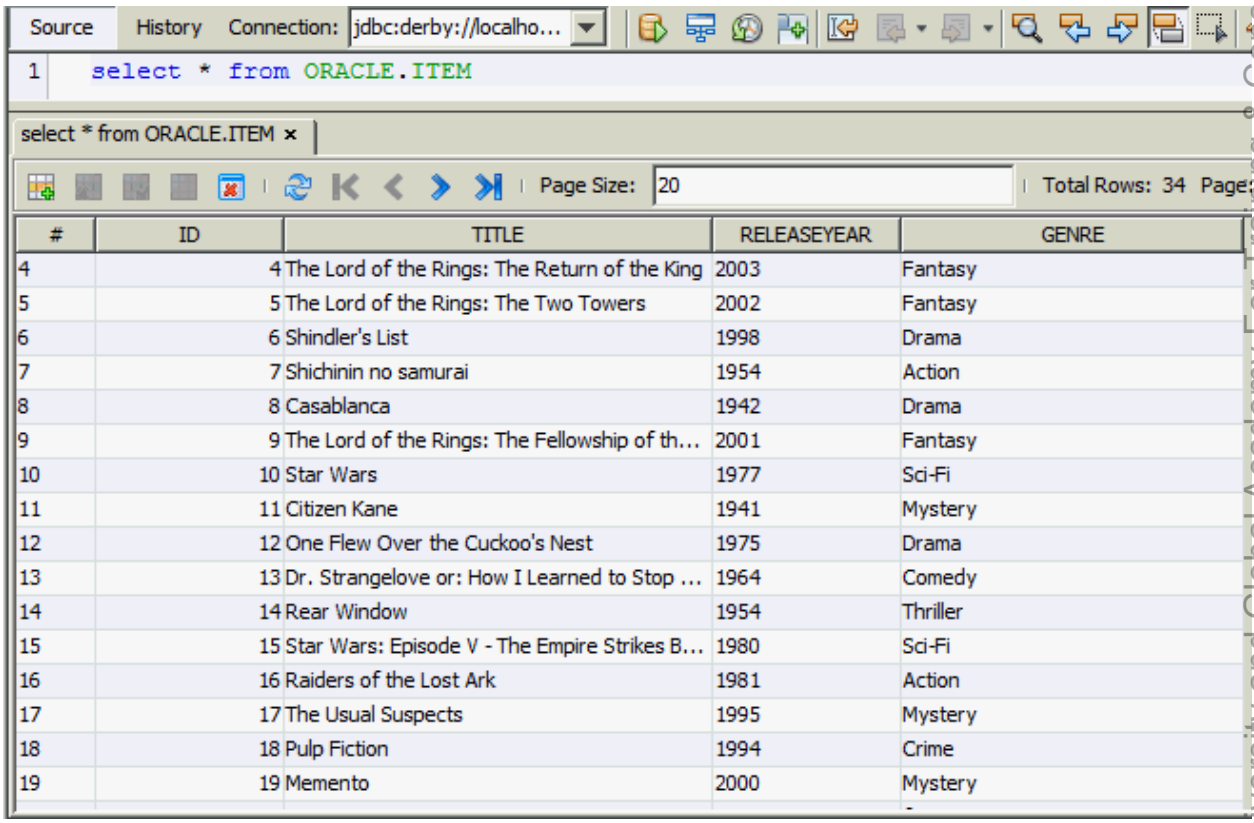
g. Execute the script.

5. To look at the data in the database, perform the following steps:

   a. Expand the new connection that you created: `jdbc:derby://localhost:1527/dvdlibrary [oracle on ORACLE]` You should see a database schema called `ORACLE`. Expand this schema.

   b. Expand Tables and you should see the table that you created with the `dvdlibrary.sql` script, `ITEM`.

   c. Right-click the `ITEM` table and select View Data to see the records that you added to this table.




| # | ID | TITLE | RELEASEYEAR | GENRE |
|---|---|---|---|---|
| 4 | 4 | The Lord of the Rings: The Return of the King | 2003 | Fantasy |
| 5 | 5 | The Lord of the Rings: The Two Towers | 2002 | Fantasy |
| 6 | 6 | Shindler's List | 1998 | Drama |
| 7 | 7 | Shichinin no samurai | 1954 | Action |
| 8 | 8 | Casablanca | 1942 | Drama |
| 9 | 9 | The Lord of the Rings: The Fellowship of th... | 2001 | Fantasy |
| 10 | 10 | Star Wars | 1977 | Sci-Fi |
| 11 | 11 | Citizen Kane | 1941 | Mystery |
| 12 | 12 | One Flew Over the Cuckoo's Nest | 1975 | Drama |
| 13 | 13 | Dr. Strangelove or: How I Learned to Stop ... | 1964 | Comedy |
| 14 | 14 | Rear Window | 1954 | Thriller |
| 15 | 15 | Star Wars: Episode V - The Empire Strikes B... | 1980 | Sci-Fi |
| 16 | 16 | Raiders of the Lost Ark | 1981 | Action |
| 17 | 17 | The Usual Suspects | 1995 | Mystery |
| 18 | 18 | Pulp Fiction | 1994 | Crime |
| 19 | 19 | Memento | 2000 | Mystery |

6. Create Entity Class. This class is needed to map the Item table to a Java class in your project in order to use JPA in the application. You can generate the class using NetBeans dialog box.

   a. Click the **Projects** tab.

   b. Right-click the **`DVDLibrary`** project and select **New**. Then select **Other** from the context menu.

   c. Select **Persistence** from the Categories pane and **Entity Classes from Database** from the File Types pane, and then click **Next**.

d. From the **Data Source** drop down, select **New Data Source…** In the Create Data Source dialog box, enter the **JNDI name** as `jdbc/dvdlibrary`. Select the **Database Connection** `as jdbc:derby://localhost:1527/dvdlibrary {ORACLE on oracle]`

e. Select `ITEM` from the **Available Tables** area of the New Entity Classes from Database dialog box and click Add. Click **Next.**

f. In the **Entity Classes** dialog box, enter `com.example.entities` in the **Package** text box and click Next.

g. Click **Finish** in the **Mapping Options** dialog box.

h. Verify if `Item.java` is created in the `com. example.entities` package in the Source Packages node of **DVDLibrary** project

i. Review the `Item.java` file that is an Entity class representing the Item table.

j. Verify if a file `persistence.xml` is created and stored in the **Configuration Files** node.

7. Create the `ItemEJB` class that would use JPA to perform the database operations.

   a. Click **Window** on the menu bar.

   b. Select **Favorites** from the context menu.

      A new Favorites tab will appear.

   c. Right-click in the **Favorites** tab and select **Add to Favorites**.

   d. Navigate to: `D:\labs\resources`.

   e. Click **Add**.

   You can close the Files window to save real estate by clicking the Files tab and pressing Ctrl+W to close the window.

   f. Copy the `ItemEJB` class from `D:\labs\resources\jpa` (in Favorites).

   g. On the Favorites tab, expand the `jpa` folder.

   h. Select `ItemEJB.java.`

   i. Right-click and select **Copy**.

   j. Select the Projects Tab. Right-click the package **`com.example.beans`** package

   k. Select **Paste**.

   l. Open the `ItemEJB` class and review the code..

      The implementation uses the `@Persistence` annotation to reference the persistence unit. This annotation works by virtue of the dependence injection capability.

      Save the `ItemEJB` class.

8. Create a new package, `com.example.exceptions` in the Source Packages node. Copy the `ItemException.java` and `PreexisitingEntityException.java` from the `D:\labs\resources\jpa` folder to the, `com.example.exceptions` package.

# Practice 4-5: Creating the `DVDLibraryBean` Class to Implement the Add DVD Feature

## Overview

In this practice, you add classes to the `DVDLibrary` project to access the DVD Library database and make changes to the `DVDLibraryBean` to enable the user to add titles to the DVD Library.

For simplicity, an `ItemEJB` class is provided to you as a session bean that contains the database operations as its business methods. `DVDLibraryBean` will use this bean to implement the add DVD feature.

## Assumptions

NetBeans is running and you have completed all the previous practices.

## Tasks

1. Create the `DVDLibraryBean` class that implements the `Serializable` interface.

   a. Add three new properties to the `DVDLibraryBean` in the `com.example.beans` package:

   ```
   private String title = "";
   private Long releaseyear;
   private String genre = "";
   ```

   The variable year is of type `Long` because numeric validation is part of future practices.

   b. Add the public getter and setter methods for these bean properties.

   c. Add the `@Named("dvd")` and the `@SessionScoped` annotations.

   ```
   @Named("dvd")
   @SessionScoped
   public class DVDLibraryBean implements Serializable {
   ```

   d. Use the `@Inject` annotation to declare `ItemEJB` object, `itemBean.`

   ```
   @Inject
       ItemEJB itembean;
   ```

   e. Declare an instance of `Item` class which is the Entity class.

   ```
     private Item item;
   ```

   f. Be sure to add the appropriate import statements:

   ```
   import javax.inject.Inject;
   ```

   **Note:** NetBeans provides a simple way to add import statements. Right-click in the editor window and select **Fix Imports**.

g. Add a method, `addDVD`, which uses the method in `ItemEJB` to add the title, year, and genre into the data store. When stored, the method should reset the title and genre properties to empty strings so that the values no longer appear in the `inputText` field on the Add DVD page.

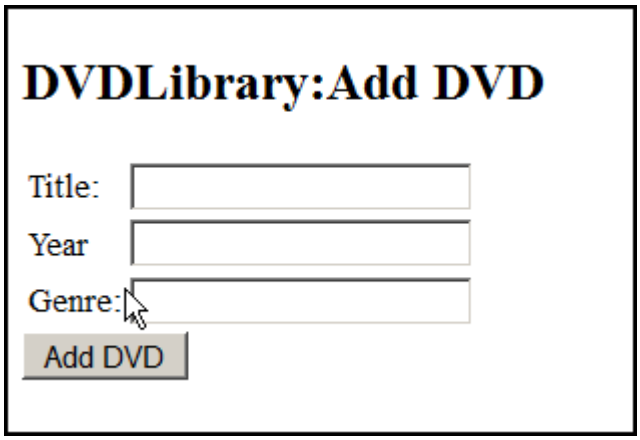
```
public String addDVD() throws PreexistingEntityException,
Exception {
       item=new Item(itembean.count()+1,title,
releaseyear.toString(), genre);
       System.out.println("    count:  "+itembean.count());
       itembean.addItem(item);
       title = "";
        genre = "";
        return "index";

 }
```

## Practice 4-6: Create the `add.xhtml` Page

### Overview

In this practice, you create the `add.xhtml` page, which implements the interface required to add a DVD to the `DVDLibrary` application.

The Add DVD page will look as shown in the following figure:




### Assumptions

Netbeans is running and you have completed all the previous practices.

### Tasks

1. Create a JSF page named `add.xhtml` that is similar to the other pages you have created.
   a. Add a title tag as you did with the login and index pages.
   b. Add a header title for the page by using the `<h2>` tag as you did before, similar to the following:

```
<h:head>
    <title>DVD Library Application</title>
</h:head>
<h:body>
    <h2>DVDLibrary:Add DVD</h2>
```

2. Add a form element that contains the following components:
   a. Use a two-column `panelGrid` element to lay out the following elements:

| | |
|---|---|
| Title: | An input field bound to the `title` property of the `dvd` managed bean |
| Year: | An input field bound to the `year` property of the `dvd` managed bean |
| Genre: | An input field bound to the `genre` property of the `dvd` managed bean |

   b. Add a `commandButton` element and invoke the addDVD method in the action attribute.

c. The result of the form element should be similar to the following:

```
<h:form>
   <h:panelGrid columns="2">
      Title: <h:inputText id="title" value="#{dvd.title}"/>
      Year <h:inputText id="year" value="#{dvd.releaseyear}"/>
      Genre: <h:inputText id="genre" value="#{dvd.genre}"/>
   </h:panelGrid>
   <h:commandButton value="Add DVD" action="#{dvd.addDVD}"/>
</h:form>
```

d. Save the `add.xhtml` page.

3. Clean and Build the project. Deploy the `DVDLibrary` application.
4. Access the URL for the login page:

   `http://localhost:7001/DVDLibrary/faces/login.xhtml`
5. Enter an arbitrary username and password combination, and click **Login**.
6. Access the URL for the `add.xhtml` page:

   `http://localhost:7001/DVDLibrary/faces/add.xhtml`
7. Enter the title, year, and genre of the DVD, and click the **Add** button.

   **Note:** Valid genres are Drama, Fantasy, Sci-Fi, Comedy, Mystery, and Action. A valid genre is required to add a DVD, and all the fields: title, year and genre, are required.
8. `index.html` opens up in the web browser window.
9. Connect to the `dvdlibrary` database.
10. Run the following query to verify that the DVD you specified in the `add.xhtml` page was successfully added to the database:

    `select title from item`

**Note:** When you run the query to verify the result in the NetBeans IDE, the DVD item that you added may not appear on the first page of the query result panel. By default, the NetBeans IDE displays 20 rows at a time. You can check the results on the following pages, or use an `order by` clause in the query, for example:

`select title from item order by id desc`

# Practices for Lesson 5: Working with Navigation

**Chapter 5**

# Practices for Lesson 5: Overview

## Practices Overview

In these practices, you will create additional JSF pages to complete the structure of the `DVDLibrary` application, use implicit navigation to move between pages, and then define navigation rules and conditional rules to fine-tune navigation in the `DVDLibrary` application.

# Practice 5-1: Creating Additional JSF Pages

## Overview

In this practice, you add the remaining pages for the `DVDLibrary` application.

## Assumptions

NetBeans is running.

## Tasks

1. Add two more JSF pages, `list.xhtml` and `prefs.xhtml`.
2. Make the pages consistent with the `index, add,` and `login` pages by adding the appropriate title and header information.

   The `list.xhtml` page will be used to display the DVD collection, and the `prefs.xhtml` page will be used to specify what information you want to display. You will design these two pages in the lesson titled "Working with Data Tables." In this practice, you will merely set up the navigation model of the application.
3. Add a form element to `list.xhtml` and `prefs.xhtml`, and inside the form element, add a `commandLink` element to enable you to go back to a home page:

   ```
   <h:form>
      <h:commandLink action="home">Back Home</h:commandLink>
   </h:form>
   ```

   Note that you are not hard-coding the `index.xhtml` page, but rather using a conceptual "home" page. Not hard coding the `index.xhtml` page gives you greater flexibility in defining the "home" page for the application.

4. Edit the `commandLink` elements in the form element of the `index.xhtml` page to enable the page flow to go from `index.xhtml` to `list.xhtml`, to `add.xhtml`, and to `prefs.xhtml`, respectively:

```
<h:form>
   <h:commandLink action="list">
      Display my DVD Library
   </h:commandLink>
   <p/>
   <h:commandLink action="add">
      Add a DVD to my library
   </h:commandLink>
   <p/>
   <h:commandLink action="prefs">
      Set Preferences
   </h:commandLink>
   <hr/>
   <h:commandLink action="login">
      Login
   </h:commandLink>
</h:form>
```

5. Run the `DVDLibrary` application.

6. Because you have not yet declared what the “home” page is, although you can navigate to the other pages, you cannot return “home.” The links should fail with a message similar to the following:

```
Unable to find matching navigation case with from-view-id
'/list.xhtml' for action 'home' with outcome 'home'
```

# Practice 5-2: Configuring Navigation Rules

## Overview

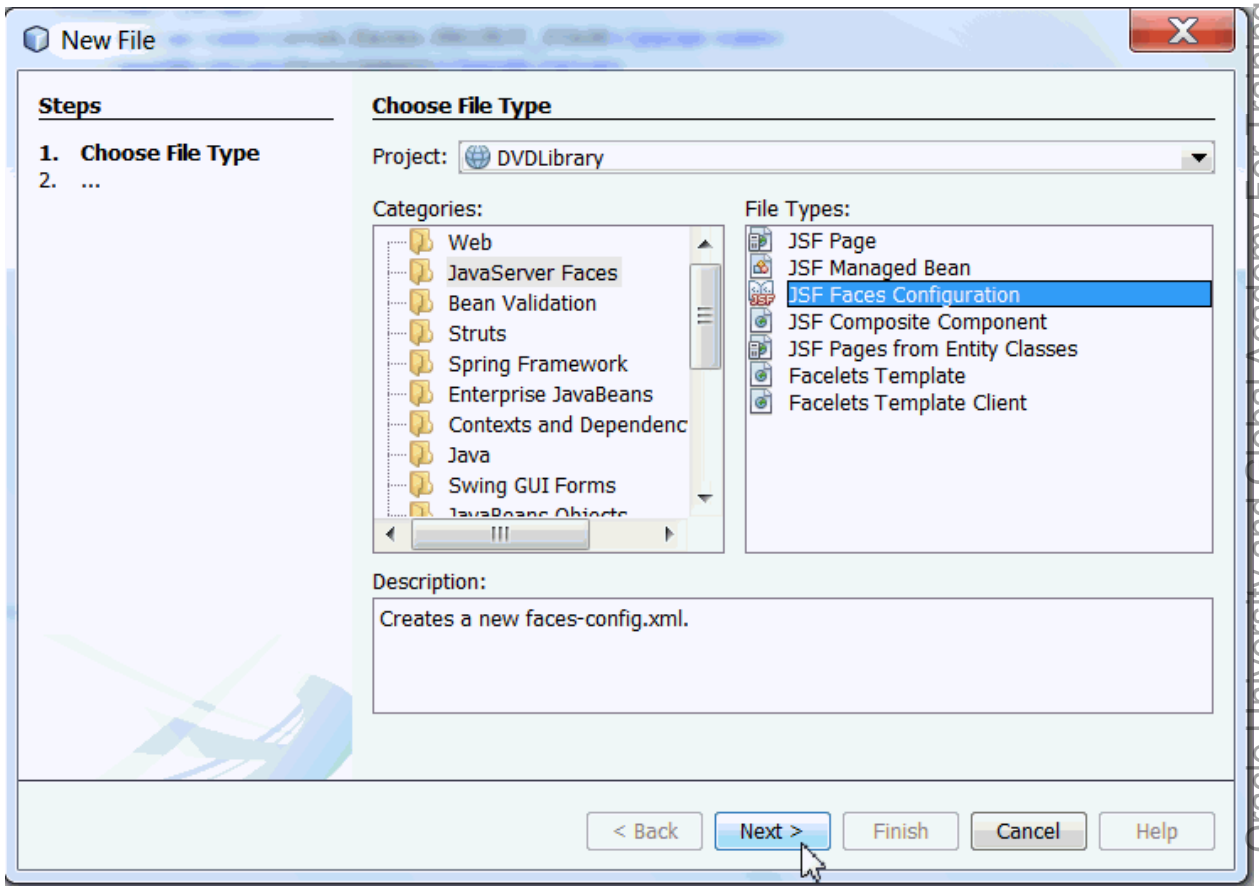
In this practice, you configure the `DVDLibrary` application to enable the page flow to go back home after adding a DVD.

## Assumptions

NetBeans is running.

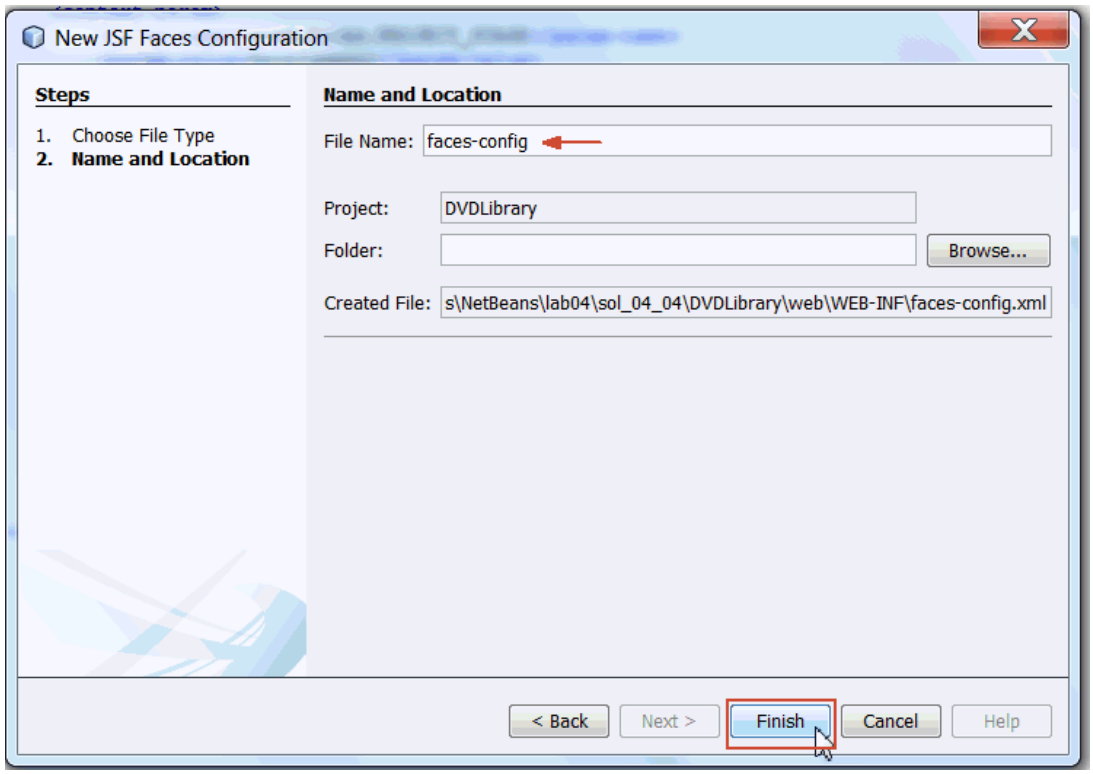
## Tasks

1. Add a JSF Faces configuration file to the `DVDLibrary` project.
   a. Right-click the **`DVDLibrary`** project icon and select **New > Other**.
   b. Select **JavaServer Faces** as the Category and **JSF Faces Configuration** as the File Type.




   c. Click **Next**.
   d. Ensure the file name is `faces-config`.

e. Click Finish.




2. Open the `faces-config.xml` file and add the following navigation rule to enable the application to go back to the home page on any action execution that results in a string outcome "home":

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

3. Run the `DVDLibrary` application.
4. Test the navigation links for `prefs` and `list`.
5. Try to add a DVD to the library.

   The page flow should go back to the home page.
6. Click **Login**, and log in with an arbitrary username and password combination.
7. Add a DVD to the collection.
8. Verify that the DVD is successfully added to the database, and the navigation goes back to the home page.

   In the following practice, you will use a conditional navigation case to force a guest user to log in when he or she tries to add a DVD.

# Practice 5-3: Using Conditional Navigation Cases

## Overview

In typical web applications, whenever a user performs an action that results in some server-side state change, such as updating a database, the user needs to be authenticated. In this practice, you implement a similar functionality that forces a guest user to log in when he or she tries to add a DVD.

## Assumptions

NetBeans is running.

## Tasks

1. Open the XML view of the `faces-config.xml` file.

   Add a new navigation rule with two navigation cases after the existing navigation rules. Both navigation cases are for the action outcome of "add." However, the first navigation case uses a condition element to check if the `username` property of the `dvd` managed bean is empty (a null value or a zero-length string). If so, the navigation moves to the login page to force the user to log in. The second navigation case navigates to the add DVD page.

```
<navigation-rule>
   <from-view-id>/*</from-view-id>
   <navigation-case>
      <from-outcome>add</from-outcome>
      <if>#{empty login.username}</if>
      <to-view-id>/login.xhtml</to-view-id>
   </navigation-case>
   <navigation-case>
      <from-outcome>add</from-outcome>
      <to-view-id>/add.xhtml</to-view-id>
   </navigation-case>
</navigation-rule>
```

2. Run the `DVDLibrary` application.
3. Verify that a guest user is forced to log in when attempting to add a DVD.

# Practice 5-4: Changing the Welcome Page

## Overview

In this practice, you change the behavior of the application startup so that the application will always require a login at the start of the application.

## Assumptions

NetBeans is running.

## Tasks

1. You successfully created a navigation rule to force the user to log in if the username property is empty, but the behavior that you want is for the application to start by prompting the user to log in.
2. The easiest way to do this is to change the page that JSF loads as the first page of the application, `welcome-file`.
   a. Expand the `Configuration Files` folder in the `DVDLibrary` project.
   b. Double-click the `web.xml` file to open it.
   c. Change the tag `welcome-file` to start with the login screen instead of the home page, the `index.xhtml` file.

```
<welcome-file-list>
   <welcome-file>faces/login.xhtml</welcome-file>
</welcome-file-list>
```

   d. Save the file.
3. Deploy and run the application. It should start by loading the login screen.

# Practices for Lesson 6: Creating Message Bundles

**Chapter 6**

# Practices for Lesson 6: Overview

## Practices Overview

In these practices, you explore the configuration file `faces-config.xml` further. You create and use a message bundle to replace the static messages and labels in the `DVDLibrary` application with text from a file that may be localized into different languages. Moving static text out of the web pages and into a message bundle makes the application text consistent, and it makes the design more flexible. For example, you can use the properties files to easily internationalize your application's messages.

# Practice 6-1: Adding the Application Message Bundle

## Overview

In this practice, you will add a message bundle properties file to the web application.

## Assumptions

NetBeans is running.

## Tasks

1. Create a new source package: `com.dvdlibrary.resources`
2. Copy the `messages.properties` and `messages_de.properties` files from the `D:\labs\resources\messages` directory to the package you created.
3. Review the `messages.properties` file.

   The `messages.properties` file contains a set of key-value pairs.

```
appName=DVD Library
dvdlibraryTitle=DVD Library Application
welcome=Welcome to the DVDLibrary Application {0}
loginBtnTxt=Login
logoutBtnTxt=Logout
passwordPrompt=Password
title_home=Home
title_login=Login
title_add=Add
title_list=List
title_prefs=Preferences
usernamePrompt=User Name
loginWelcome=Welcome {0}!
loginHeader=Please login
goToList_library=Display the DVD library
goToAdd_dvd=Add a DVD to my collection
goToSet_prefs=Set user preferences
lang=Choose your language
langEn=English
langDe=Deutsh
welcome_list=Number of DVDs in collection = {0}
column_Title=DVD Title
column_Year=Year
column_Genre=Genre
add_genre=Add a new genre
instructions_set=Columns to display in list:
btn_prefs=Update Preferences
```

```
btn_addDVD=Add DVD
btn_cancel=Cancel
addMessage=DVD added:
```

4. Review the `messages_de.properties` file.

   Note that this file contains the same keys, but the values are localized to German.

# Practice 6-2: Adding Messages to the DVDLibrary Application Pages

## Overview

In this practice, you replace the static text and labels with messages from the resource bundle.

## Assumptions

NetBeans is running.

## Tasks

1. Edit the `index.xhtml` JSF page.
   a. Open the `index.xhtml` JSF page.
   b. Add the new namespace declaration for the JSF core tag library as follows:

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

   c. Add the `loadBundle` core tag as a child of the html root element. Use this tag to load the message bundle message, and assign the message bundle to the variable `msg`.

```
<f:loadBundle basename="com.dvdlibrary.resources.messages"
var="msg"/>
```

   d. Change the value of the `title` and `h2` elements from the default static text to use the `appName` and `title_home` properties in the message bundle, for example:

```
<title>#{msg.dvdLibraryTitle}</title>
...
<h2>#{msg.appName}:#{msg.title_home}</h2>
```

   e. Use the `welcome` message in the resource bundle to replace the static welcome message.

      Because the `welcome` message is parameterized (the `{0}` notation), instead of directly referencing the property value by using an EL expression, you should use the HTML tag `outputFormat` to load the value, and use the core tag `param` to assign a value to the parameter. For example:

```
<h:outputFormat value="#{msg.welcome}">
<f:param value="#{empty login.username ? 'Guest' :
login.username}"/>
 </h:outputFormat>
```

f. Update the `commandLink` elements to use the `gotoList_library`, `gotoAdd_dvd`, `gotoSet_prefs`, and `loginBtnTxt` properties to display the text for each link:

```
<h:form>
   <h:commandLink value="#{msg.goToList_library}" action="list"/>
   <p/>
   <h:commandLink value="#{msg.goToAdd_dvd}" action="add"/>
   <p/>
   <h:commandLink value="#{msg.goToSet_prefs}" action="prefs"/>
   <hr/>
   <h:commandLink value="#{msg.loginBtnTxt}" action="login"/>
</h:form>
```

g. Save the `index.xhtml` page.

h. Deploy and run the `DVDLibrary` application to test your changes to the home page.

2. Edit the `login.xhtml` page.

a. Open the `login.xhtml` page.

b. Add the new namespace declaration for the JSF core tag library to the page.

c. Add the `loadBundle` core tag as a child of the html root element. Use this tag to load the message bundle message and assign the message bundle to the variable `msg`.

d. Change the value of the `title` and `h2` elements from the default static text to use the `appName` and `title_login` properties in the message bundle.

e. Update the form element by replacing the static text with the `usernamePrompt`, `passwordPrompt`, and `loginBtnTxt` properties:

```
#{msg.usernamePrompt}: <h:inputText value="#{login.username}"/><p/>
#{msg.passwordPrompt}: <h:inputSecret value="#{login.password}"/><p/>
<h:commandButton value="#{msg.loginBtnTxt}" action="index"/>
```

f. Save the `login.xhtml` page.

g. Deploy and run the `DVDLibrary` application to test your changes to the login page.

3. Edit `add.xhtml` page

a. Open the `add.xhtml` page.

b. Add the new namespace declaration for the JSF core tag library to the page.

c. Add the `loadBundle` core tag as a child of the html root element. Use this tag to load the message bundle message and assign the message bundle to the variable `msg`.

d. Change the value of the `title` and `h2` elements from the default static text to use the `appName` and `title_add` properties in the message bundle.

e. Replace the static text for the Title, Year, and Genre labels. You will use keys that you can also use to define the column titles when you display the library in a later practice:

```
#{msg.column_Title}: <h:inputText id="title" value="#{dvd.title}"/>
#{msg.column_Year}: <h:inputText id="year"
value="#{dvd.releaseyear}"/>
#{msg.column_Genre}: <h:inputText id="genre" value="#{dvd.genre}"/>
```

f. Replace the static text for `commandButton`:

```
<h:commandButton value="#{msg.btn_addDVD}" action="#{dvd.addDVD}"/>
```

g. Save the `add.xhtml` page.

4. Edit `list.xhtml` page.

a. Open the `list.xhtml` page.

b. Add the core tags declaration to the html tag:

```
xmlns:f="http://java.sun.com/jsf/core"
```

c. Add the `loadBundle` core tag as a child element of the html root element:

```
<f:loadBundle basename="com.dvdlibrary.resources.messages"
var="msg"/>
```

d. Replace the static text in the title tags with a message:

```
#{msg.dvdlibraryTitle}
```

e. Replace the static header text with messages:

```
#{msg.appName}:#{msg.title_list}
```

f. Save the `list.xhtml` page.

5. Edit `prefs.xhtml` page.

a. Open the `prefs.xhtml` page and repeat the preceding steps:

b. Be sure to replace the header text with the appropriate message:

```
#{msg.appName}:#{msg.title_prefs}
```

6. Deploy and run the `DVDLibrary` application to test your changes.

For now, you will not replace the static message: “Back Home” in the `list` and `prefs` pages.

# Practice 6-3: Testing the DVDLibrary Application in Multiple Languages

## Overview

In this practice, you will test your application in a different language.

## Assumptions

NetBeans is running.

## Tasks

1. Open the XML view of the `faces-config.xml` file.
2. Add an application element to the `faces-config.xml` root element and add a `locale-config` element to the application element. Specify the default locale of the application to be English and add German as a supported locale:

```
<application>
   <locale-config>
      <default-locale>en</default-locale>
      <supported-locale>de</supported-locale>
   </locale-config>
</application>
```

3. Save the `faces-config.xml` file.
4. Run the `DVDLibrary` application.

   Notice the title and heading information of the home page. The text should be displayed in English.
5. In the following steps, you configure a different preferred language setting in your web browser to verify JSF's localization feature.

   **Note:** The following steps apply to **Firefox**. If you are using a different browser, use the steps required by the browser to add language support.

   a. Select **Tools** from the web browser menu, and then select **Options** from the context menu.

   c. Click the **Content** tab. Click the **Choose** button in the Languages section at the bottom of the screen.

   d. In the Languages dialog box, click the **Select a language to add** drop-down menu and select **German (Germany) [de-de]**.

   e. Click **Add**.

   f. Click **Move Up**, to move German (Germany) [de-de] to the top of the list to make it your preferred language.

   g. Click **OK** to close the Languages dialog box.

   h. Click **OK** again to close the Options dialog box.

6. Reload the `DVDLibrary` application.
   a. Notice the title and the heading information of the home page. The text should be displayed in German.
   b. You can change the preferred language back to English in the browser.

# Practice 6-4: Loading the Resource Bundle at the Application Level

## Overview

In this practice, you add a `resource-bundle` declaration in the `faces-config.xml` file.

## Assumptions

NetBeans is running.

## Tasks

1. In the previous practices, you loaded the resource bundle `messages.properties` by using the `loadBundle` tag at the top of each page. By declaring the resource bundle globally, you can avoid this step.
2. Open the `faces-config.xml` file.
3. Add a resource bundle declaration inside the root application tag:

```
<resource-bundle>
   <base-name>com.dvdlibrary.resources.messages</base-name>
   <var>msg</var>
</resource-bundle>
```

4. Save the `faces-config.xml` file.
5. You can remove the `loadBundle` declaration from the `add.xhtml`, `index.html`, `login.xhtml, list.xhtml and prefs.xhtml` pages.

# Practices for Lesson 7: Creating Templates for the DVDLibrary Application

**Chapter 7**

# Practices for Lesson 7: Overview

## Practices Overview

In these practices, you will improve the look-and-feel of the `DVDLibrary` application by using a template. In practice, you would likely develop the template first, but in this course, you learn how simple it is to refactor an existing application by using templates.

# Practice 7-1: Applying a Template to the `DVDLibrary` Application

## Overview

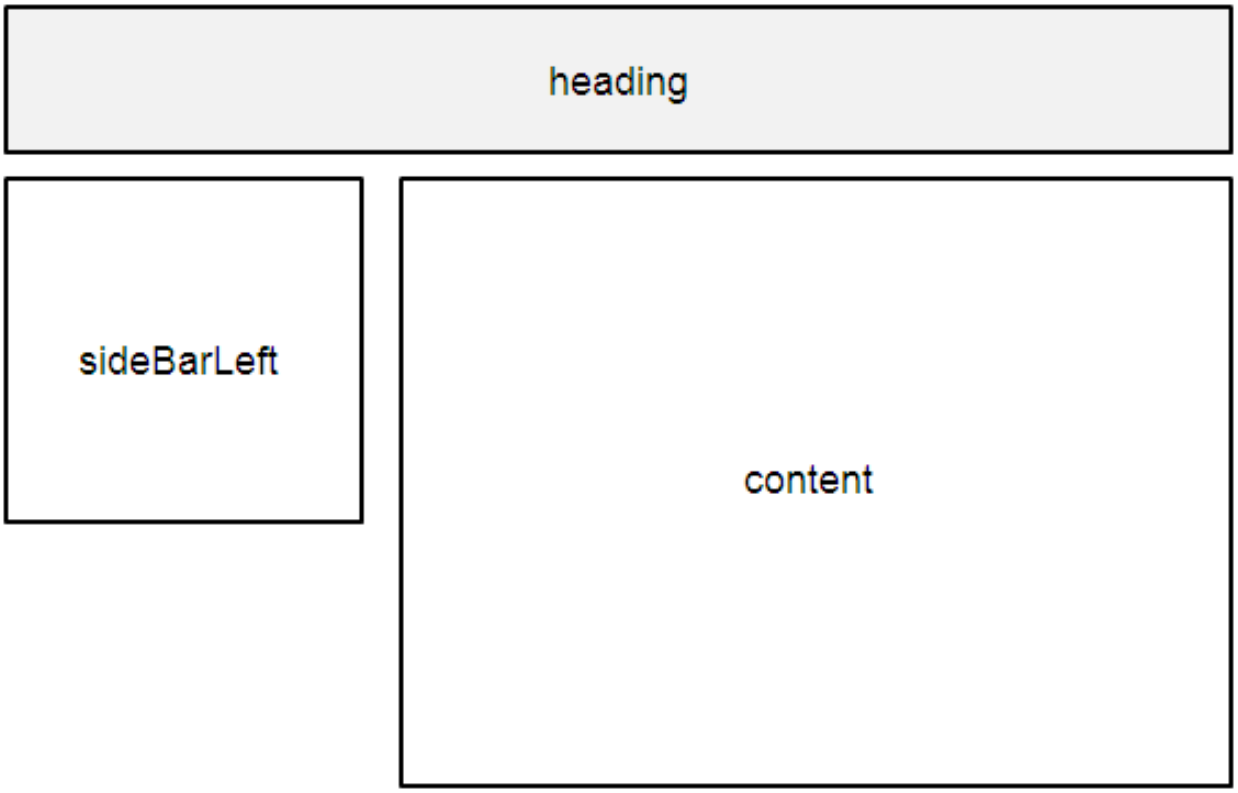
In this practice, you apply a template to the existing `DVDLibrary` application.

## Assumptions

NetBeans is open and you should have completed the practices for the previous lessons.

## Tasks

1. The `DVDLibrary` application template has three parts: a header section, where the titles will appear, a sidebar on the left, and a content area to the right of the sidebar below the header.




2. Copy the `D:\labs\resources\templates` directory to the Web Pages folder of `DVDLibrary` project. Verify if `masterLayout.xhtml` file is available in the `Web Pages\templates` folder.
3. Open the `masterLayout.xhtml` file.

4. The head tags now include a style sheet. This style sheet defines some of the characteristics of the elements used in the template, including the size and location of the components of the template font titles, and so on. You will add this style sheet to the project in a subsequent step.

   a. Notice that the title of the application is now included in the template, so you do not need to include it on every page.

```
<h:head>
   <h:outputStylesheet library="css" name="styles.css"/>
   <title>#{msg.dvdlibraryTitle}</title>
</h:head>
```

   b. In the `h:body` tag, you can see that the template defines the three parts of the template interface: a heading, sideBarLeft, and content. It uses the `<div>` tags to define the sections and the Facelets tags `ui:insert` to define what the composition pages should define to override the content specified in the `masterLayout` template.

```
<h:body>
   <!-- Header includes the Welcome message and function title -->
   <div id="heading" class="header">
      <ui:include src="/sections/dvdlibrary/headerWelcome.xhtml"/>
      <!-- The function title format = DVD Library:<Function> -->
      <ui:insert name="heading"/>
   </div>
   <!-- SideBarLeft is located below the header to the left -->
   <div id="sideBarLeft">
      <ui:insert name="sideBarLeft">
         <ui:include src="/sections/dvdlibrary/sideBarLeft.xhtml"/>
      </ui:insert>
   </div>
   <!-- Content section is right of the Side Bar below the header -->
   <div id="content">
      <ui:insert name="content"/>
   </div>
</h:body>
```

   The heading section uses a `ui:include` tag to insert a default welcome message at the top of the heading section, followed by the heading content inserted by each page defined by the `<ui:insert name="heading">` tag.

5. Copy the `D:\labs\resources\sections` directory to the Web Pages folder of `DVDLibrary` project. Verify if two folders, `dvdlibrary` and the other `login` exists. These folders will hold the specific section templates for the `DVDLibrary` application.

6. Open the `headerWelcome.xhtml` file in the `sections\dvdlibrary` folder of the `DVDLibrary` project.

   Note that the head and body tags are standard HTML page tags, instead of the usual JSF tags. This is because the `ui:composition` tag causes everything outside to be ignored. This file could have been defined with only the content inside the `ui:composition` tag, but by including the other tags, you get the benefit of the syntax checker of the IDE. The header defines the layout of the area at the top of the web page, the `<div class="header">` tag, and includes the default welcome message that you created earlier.

```
<head><title>IGNORED</title></head>
<body>
   <ui:composition>
      <div class="header">
         <h:outputFormat value="#{msg.welcome}">
            <f:param value="#{empty dvd.username ? 'Guest' :
dvd.username}"/>
         </h:outputFormat>
         !
      </div>
   </ui:composition>
</body>
```

7. Open the `sideBarLeft.xhtml` file in the `sections\dvdlibrary` folder of the DVDLibrary project.

   This composition element is included in the `masterLayout` template and inserted into the `div sideBarLeft` section of the web page.

```
<ui:composition>
   <h:form>
      <hr/>
      <h:commandLink value="#{msg.goToList_library}" action="list"/>
      <p/>
      <h:commandLink value="#{msg.goToAdd_dvd}" action="add"/>
      <p/>
      <h:commandLink value="#{msg.goToSet_prefs}" action="prefs"/>
      <hr/>
      <h:commandLink value="#{msg.loginBtnTxt}" action="login"/>
    </h:form>
</ui:composition>
```

8. Open the `sideBarLeft.xhtml` file in the `sections\login` folder of the `DVDLibrary` project.

   This `sideBarLeft.xhtml` file will be used to override the `sideBarLeft.xhtml` file defined by the `masterLayout.xhtml` template. The login sidebar includes a reference to a graphic image instead of the menu that the application will have—so that until the users log in, they do not see the functionality of the application such as list, add, and so on.

```
<ui:composition>
   <div class="welcome">
      <div class="welcomeImage">
         <h:graphicImage library="images" name="dvd.gif"/>
      </div>
   </div>
</ui:composition>
```

9. Add the Style Sheet to the `DVDLibrary` project.
   a. Create a folder under the Web Pages folder called `resources`.
   b. Create a folder under the resources folder called `css`.
   c. Copy the `styles.css` file from the `D:\labs\resources\css` directory.
   d. Paste the `styles.css` file into the newly created `resources\css` folder in the project.
10. Add the graphic image to the `DVDLibrary` project.
   a. Create a `folder` called images under the `resources` folder in the `DVDLibrary` project.
   b. Copy the `dvd.gif` image from the `D:\labs\resources\images` folder.
   c. Paste the `dvd.gif` image file into the `resources\images` folder in the `DVDLibrary` project.

# Practice 7-2: Refactoring the DVDLibrary Pages to Use the Template

## Overview

In this practice, you apply the template to the pages of the `DVDLibrary` application.

## Assumptions

## Tasks

1. Open the `login.xhtml` page.
2. Modify the page to use the template by changing the following:
   a. Verify if the following declaration is added to the html tag:

      ```
      xmlns:ui="http://java.sun.com/jsf/facelets"
      ```

   b. Remove the JSF head and body tags, because these are ignored outside the composition tag anyway:

      ```
      <h:head>
         <title>#{msg.dvdlibraryTitle}</title>
      </h:head>
      <h:body>
         <h2>#{msg.appName}:#{msg.title_login}</h2>
      ```

   c. Replace them with the following, which indicates that this page should use the `masterLayout` template.

      ```
      <head><title>IGNORED</title></head>
      <body>
         <ui:composition template="/templates/masterLayout.xhtml">
            <ui:define name="heading">
               #{msg.appName}:#{msg.loginHeader}
            </ui:define>

            <ui:define name="sideBarLeft">
               <ui:include src="/sections/login/sideBarLeft.xhtml"/>
            </ui:define>
      ```

d. Wrap the existing JSF with a `ui:define` tag, similar to the following:

```
      <ui:define name="content">
         <h:form>
            <h:panelGrid columns="2">
            #{msg.usernamePrompt}:
            <h:inputText value="#{login.username}"/>
            <p/>
            #{msg.passwordPrompt}:
            <h:inputSecret value="#{login.password}"/>
            <p/>
            <h:commandButton value="#{msg.loginBtnTxt}"
action="home"/>
         </h:panelGrid>
```

```
      </h:form>
    </ui:define>
```

e. Make sure that you close the `<ui:composition>` and `<body>` tags:

```
      </ui:composition>
   </body>
</html>
```

f. Save the `login.xhtml` page.

3. Open the `add.xhtml` file.

a. Verify if the following declaration is added to the `html` tag:

```
xmlns:ui="http://java.sun.com/jsf/facelets"
```

b. Replace this section of the page:

```
<h:head>
   <title>#{msg.dvdlibraryTitle}</title>
</h:head>
<h:body>
   <h2>#{msg.appName}:#{msg.title_add}</h2>
```

c. With the following:

```
<head><title>IGNORED</title></head>
<body>
   <ui:composition template="/templates/masterLayout.xhtml">
      <ui:define name="heading">
         #{msg.appName}:#{msg.title_add}
      </ui:define>
```

Note that other than the login page, you use the standard `sideBarLeft` section that includes the menu. Because the `masterLayout` template includes this automatically, you do not need to define a sidebar of your own.

d. Wrap the `<h:form>` content with the appropriate JSF composition tag for the content section:

```
      <ui:define name="content">
         <h:form>
            <h:panelGrid columns="2">
               #{msg.column_Title}:
               <h:inputText id="title" value="#{dvd.title}"/>
               #{msg.column_Year}:
               <h:inputText id="year"
value="#{dvd.releaseyear}"/>
               #{msg.column_Genre}:
               <h:inputText id="genre" value="#{dvd.genre}"/>
           </h:panelGrid>
           <h:commandButton value="#{msg.btn_addDVD}"
action="#{dvd.addDVD}"/>
        </h:form>
     </ui:define>
```

e. Close the `<ui:composition>` and `<body>` tags:

```
     </ui:composition>
   </body>
</html>
```

f. Save the `add.xhtml` page.

4. Apply the template process to the `list` and `prefs` pages—for these, the content section is still a simple return link:

```
<ui:define name="content">
   <h:form>
       <h:commandLink action="home">Back Home</h:commandLink>
   </h:form>
</ui:define>
```

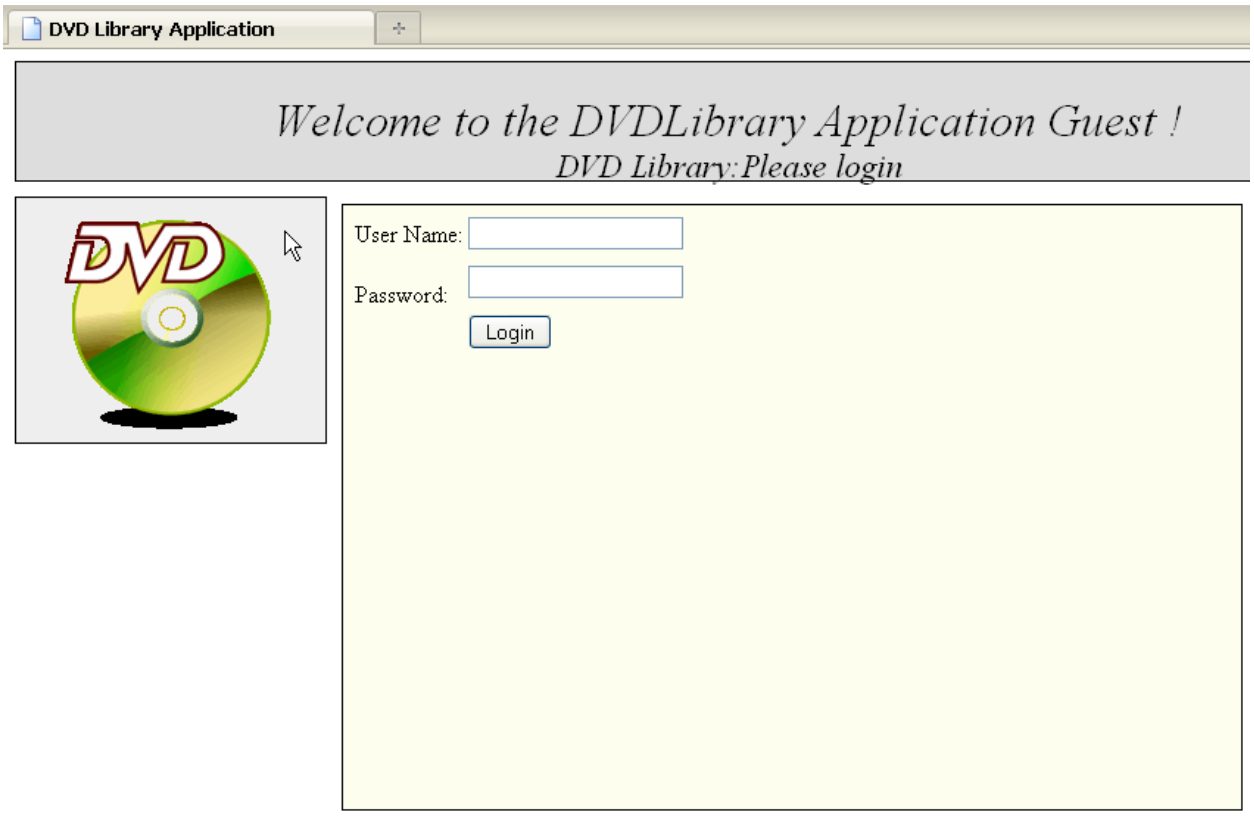
5. Because the `masterLayout.xhtml` template now includes the command links in the sidebar, you do not need the `index.xhtml` any longer.

   a. Open the `faces-config.xml` file.

   b. Change the “home” location to open the list page as the default view:

```
<from-outcome>home</from-outcome>
<to-view-id>/list.xhtml</to-view-id>
```

6. Build, deploy, and run the application. Your new DVDLibrary application should look similar to the following:




7. Test the application to make sure that the navigation works.

# Practices for Lesson 8: Validating and Converting Data

**Chapter 8**

# Practices for Lesson 8: Overview

## Practices Overview

In these practices, you will add data validation and conversion to the “Add DVD” feature of the `DVDLibrary` project.

# Practice 8-1: Adding a Data Conversion Error Message

## Overview

In this practice, you will add an error message to the “Add DVD” feature.

## Assumptions

## Tasks

1. Run the `DVDLibrary` application.
2. Try to log in, and then add a DVD. When you enter the value for the releaseyear, enter some non-numerical numbers such as “Last Year.”
3. When you submit the form, an error message is displayed as follows:

   ```
   j_idt25:year: 'Last Year' must be a number consisting of one or
   more digits.
   ```

4. The error message is due to the fact that the `releaseyear` property of the `dvd` managed bean has a type of long. JSF run time applies the default converter, `javax.faces.converter.LongConverter`, to the input value bound to the property. Because it cannot convert the string “Last Year” into a valid long value, the JSF implementation displays the default conversion error message for the associated converter.
5. Complete the following steps to reorganize the error message so that it appears next to the relevant UI component:
   a. Modify `add.xhtml` by setting the columns attribute of the `panelGrid` tag to 3.
   b. After each UI component tag, add a `message` tag:

```
<h:form>
   <h:panelGrid columns="3">
      #{msg.column_Title}
      <h:inputText id="title" value="#{dvd.title}"/>
      <h:message for="title" />
      #{msg.column_Year}
      <h:inputText id="year" value="#{dvd.releaseyear}"/>
      <h:message for="year"/>
      #{msg.column_Genre}
      <h:inputText id="genre" value="#{dvd.genre}"/>
      <h:message for="genre"/>
   </h:panelGrid>
   <h:commandButton value="#{msg.btn_addDVD}"
                    action="#{dvd.addDVD}"/>
</h:form>
```

6. Make the error messages more noticeable by changing the text color.
   a. The CSS that you added earlier has a style class called "`error`." Add it to each error message:

```
<h:message for="title" styleClass="error"/>
```

7. Save `add.xhtml` and retest the application.
8. You will see that the error message is still a little less than obvious:

```
j_idt25:year: 'Last Year' must be a number between -
9223372036854775808 to 9223372036854775807 Example: 98765432
```

9. Fortunately, there are two options that you can add to the message tag that help:
   – `showSummary="true"`: Provides a summary of the error message that is more human-readable
   – `ShowDetail="false"`: Turns off the default behavior of providing the detailed error message
10. Add the `showSummary` and `showDetail` elements to the error messages:

```
<h:message for="year" styleClass="error" showSummary="true"
showDetail="false"/>
```

11. Save and retest. The error message should be much easier to understand now:

```
j_idt25:year: 'Last Year' must be a number consisting of one or
more digits.
```

# Practice 8-2: Using a Non–default Data Converter

## Overview

In this practice, you will configure an explicit number converter for the year field, instead of using the default converter applied to the field based on the property type (`Long`).

## Assumptions

NetBeans is running and you have completed the previous practices.

## Tasks

1. Open the `add.xhtml` page if necessary.
2. Add the following to the tag library declarations:

   ```
   xmlns:f="http://java.sun.com/jsf/core"
   ```

3. Update the input tag for the year field by adding an `f:convertNumber` as the child element, and set its `integerOnly` attribute to `true`:

   ```
   <h:inputText id="year" value="#{dvd.releaseyear}">
      <f:convertNumber integerOnly="true" groupingUsed="false"/>
   </h:inputText>
   ```

4. Repeat the preceding steps and verify that the error message is different than before. For example, when you enter `Last Year`, the data conversion error message should be as follows:

   ```
   j_idt25:year: 'Last Year' is not a number.
   ```

# Practice 8-3: Skipping Data Conversion on Cancel

## Overview

In this practice, you add a Cancel button to the “Add DVD” feature to enable the user to cancel out of the add action and prevent validation from occurring.

## Assumptions

NetBeans is running and you have completed the previous practices.

## Tasks

1. Edit the `add.xhtml` page.
2. Add a Cancel button to the form by setting the action attribute to home and the `immediate` attribute to true:

```
<h:commandButton value="#{msg.btn_cancel}" action="home"
immediate="true"/>
```

3. Save `add.xhtml` and run the application again. After entering an invalid year, when you click Cancel, the navigation should immediately go back to the home page, bypassing the data conversion and validation.

# Practice 8-4: Making Input Parameters Mandatory

## Overview

In this practice, you make the parameters mandatory, by using the built-in validator, `required`.

## Assumptions

NetBeans is running and you have completed the previous practices.

## Tasks

1. Modify `add.xhtml` by setting the `required` attribute of each input component to true:

```
<h:inputText id="title" value="#{dvd.title}" required="true"/>
```

2. Run the `DVDLibrary` application.
3. Try to add a DVD by leaving one or all the fields empty.
4. When you submit the form, you should see the following error message:

```
j_idt3:title: Validation Error: Value is required.
j_idt3:year: Validation Error: Value is required.
j_idt3:genre: Validation Error: Value is required
```

# Practice 8-5: Validating the Year Range

## Overview

In this practice, you use a standard validator to ensure that the year input is in the range between two numbers.

## Assumptions

NetBeans is running and you have completed the previous practices.

## Tasks

1. Modify the `add.xhtml` page by adding a validator to the year field that restricts the year value to be between 1909 and 2013.

```
<h:inputText id="year" value="#{dvd.releaseyear}"
required="true">
   <f:convertNumber integerOnly="true" groupingUsed="false"/>
   <f:validateLongRange minimum="1909" maximum="2013"/>
</h:inputText>
```

2. Run the `DVDLibrary` application.
3. Try to add a DVD by providing a year value that is out of the specified range.
4. When you submit the form, you should see the following error message:

```
j_idt3:year: Validation Error: Specified attribute is not
between the expected values of 1,909 and 2,013.
```

# Practice 8-6: Using EL Expressions in Validators

## Overview

In this practice, you use the Java `Calendar` object to calculate the current year dynamically, and use the value to specify the validation range for the `year` property. This requires a new method to be added to the `dvd` managed bean.

## Assumptions

NetBeans is running and you have completed the previous practices.

## Tasks

1. Open the managed bean class `DVDLibraryBean` and add a `currentYear` method to calculate the current year by using the `Calendar` object:

```
public Long getCurrentYear() {
   Calendar rightNow = Calendar.getInstance();
   Long x = new Long( rightNow.get(Calendar.YEAR));
   return x;
}
```

2. On the `add.xhtml` page, modify the validator to use the EL expression `#{dvd.currentYear}` to specify the maximum value in the range:

```
<h:inputText id="year" value="#{dvd.releaseyear}"
required="true">
   <f:convertNumber integerOnly="true" groupingUsed="false"/>
   <f:validateLongRange minimum="1909"
maximum="#{dvd.currentYear}"/>
</h:inputText>
```

3. Run the `DVDLibrary` application.
4. Try to add a DVD by providing a year value that is out of the specified range.
5. When you submit the form, you should see an error message that is similar to the one in the previous task.

# Practice 8-7: Using Custom Validation and Conversion Messages

## Overview

In this practice, you add a message bundle to override some of the default error messages provided by the default JSF implementation.

## Assumptions

NetBeans is running and you have completed the previous practices.

## Tasks

1. Create a property file `errors.properties` in the source package, `com.dvdlibrary.resources` (where the other message bundles are located), of the `DVDLibrary` project.
2. Add the following properties:

```
javax.faces.converter.NumberConverter.NUMBER={2}: ''{0}'' please
enter a number
javax.faces.validator.LongRangeValidator.NOT_IN_RANGE={2}:
Please enter a numeric value MUST between {0} and {1}
```

3. Open the XML view of the `faces-config.xml` file.
4. Add a `message-bundle` element to the application element, and specify errors as the content of the `message-bundle` element:

```
<application>
   <message-bundle>com.dvdlibrary.resources.errors</message-
bundle>
...
</application>
```

5. Run the application. Try to enter a non-numeric value in the year field and verify that the new converter error message is displayed.
6. Run the application again. Try to enter a number that is out of range and verify that the new validation error message is displayed.

# Practice 8-8: Selecting a Genre from a Pull-Down List

## Overview

In this practice, you add an HTML `SelectOneMenu` component to `add.xhtml`. The `SelectOneMenu` data is a list of all movie genres from the DVD data. This allows the user to select a genre from those that are already available in the database. The model object, `DVDLibraryDAO`, supports retrieving genres as a `List` object. The `DVDLibraryBean` exposes that feature with the `getGenreList` method.

The Add form will look as shown in the following figure:


## Assumptions

NetBeans is running and you have completed the previous practices.

## Tasks

1. Add the functionality in the `dvd` managed bean for retrieving the list of genres.

   Add an instance variable, `genreList`, to hold a reference to the list of genres:

   ```
   private ArrayList genreList = null;
   ```

   Add a getter method for `genreList` that uses a `SelectItem`, which is a JSF object. This object is accessed in `add.xhtml`.

   ```
   public ArrayList getGenreList() {
           if (genreList == null) {
               genreList = new ArrayList<String>();
               List freshGenres = itembean.getGenres();
               Iterator g = freshGenres.iterator();
               while (g.hasNext()) {
                   String item = (String) g.next();
                   SelectItem n = new SelectItem(item, item);
                   genreList.add(n);
               }
           }
   ```

```
        return genreList;
}
```

Make sure that you update the import statements to include the new classes used in the code. You can type Ctrl+Shift+i to automatically update the import statement list.

2. Replace the Genre input text field with a `SelectOneMenu` component in `add.xhtml` with a `SelectOneMenu` component that uses a `selectItems` tag to dynamically load menu choices from `dvd.genreList`.

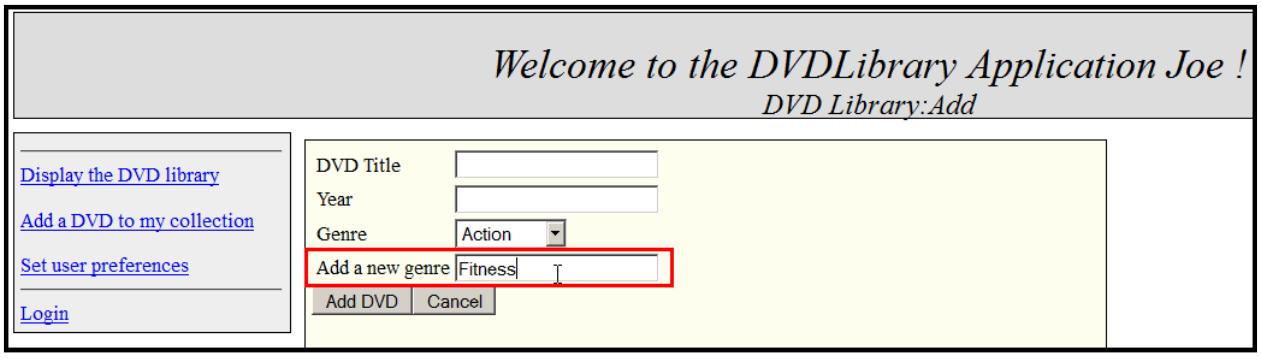
```
#{msg.column_Genre}
<h:selectOneMenu id="genre" value="#{dvd.genre}">
   <f:selectItems value="#{dvd.genreList}" />
</h:selectOneMenu>
```

# Practice 8-9: Adding a New Genre

## Overview

In this practice, you enhance the "Add DVD" feature by adding the ability to add a new genre.

The Add form will look as shown in the following figure:




## Assumptions

NetBeans is running and you have completed the previous practices.

## Tasks

1. Add a `newGenre` property to `DVDLibraryBean.java`.

```
private String newGenre;
```

2. Create a new getter and setter method for the `newGenre` property:

```
public String getNewGenre() {
   return newGenre;
}

public void setNewGenre(String newGenre) {
   this.newGenre = newGenre;
}
```

3. Add an `addGenre` method. This method first checks if the current `genreList` already has a `SelectItem` with the same value. If so, the value will not be added. Otherwise, a new `SelectItem` is created, and then added to the `genreList`.

```
public void addGenre(String newGenre) {
   if ( newGenre.equals(""))
   return;
   Iterator g = genreList.iterator();
   while (g.hasNext()) {
      SelectItem item = (SelectItem)g.next();
      if (item.getValue().equals(newGenre))
         return;
      }
   SelectItem newItem = new SelectItem(newGenre, newGenre);
   genreList.add(newItem);
}
```

4. Modify the `addDVD` method so that it uses the value from `newGenre`, when one is available.

```
public String addDVD() {
   if (!newGenre.equals("")) {
      addGenre(newGenre); // add to cache
      genre = newGenre;
   }

   DVDItem item = new DVDItem(title, year.toString(), genre);
   boolean result = dao.addDVD(username, item);
   title = "";
   genre = "";
   year = getCurrentYear();
   setNewGenre("");
   return "home";
}
```

5. Add a new row to `panelGrid` of the `add.xhtml` page with the following components:
   a. The `add_genre` property of the `msg` resource bundle
   c. An `inputText` field bound to the `newGenre` property of the `dvd` managed bean
   d. An `h:message` component for the `newGenre` input field

```
#{msg.add_genre}
<h:inputText id="newGenre" value="#{dvd.newGenre}" />
<h:message for="newGenre" styleClass="error" showSummary="true"
showDetail="false"/>
```

6. Run the `DVDLibrary` application to verify the new functionality.

# Practices for Lesson 9: Working with Data Tables

**Chapter 9**

# Practices for Lesson 9: Overview

## Practices Overview

In these practices, you will add functionality to the `DVDLibrary` project to display the list of DVD titles present in the `Items` table.

# Practice 9-1: Updating the `DVDLibraryBean` Class and Designing the `list.xhtml` Page

## Overview

In this practice, you update the `DVDLibraryBean` class to implement the methods required to hold and retrieve a `java.util.List` of `DVDItems` present in the `Items` table.

## Assumptions

NetBeans is running and you have successfully completed the practices of the previous lesson.

## Tasks

1. Modify the `DVDLibraryBean` class as follows:

   a. Add an instance variable, `dvdCollection`, to hold a reference to the `List` of the `DVDItems`.

   ```
   private List dvdCollection;
   ```

   b. Add a getter method, `getDVDCollection`, which returns a reference to `dvdCollection`. If `dvdCollection` is null, initialize `dvdCollection` with a call to `itembean.getAllItems()`.

   ```
   public List getDvdCollection() {
        if ( this.dvdCollection == null ) {
      this.dvdCollection = itembean.getAllItems();
   }
   return dvdCollection;
}
   ```

   c. Add a method `getSize` to return the number of items in `dvdCollection`.

   ```
   public int getSize() {
     return getDVDCollection().size();
   }
   ```

2. Save the `DVDLibraryBean` class.
3. Open the `list.xhtml` page.
4. Under the level-2 heading, remove:

   ```
   #{msg.appName}:#{msg.title_list}
   ```

5. Replace it with the `welcome_list` message in the `msg` resource bundle, and use the EL expression `#{dvd.size}` to populate the parameter in the message:

```
<h:outputFormat value="#{msg.welcome_list}">
   <f:param value="#{dvd.size}"/>
</h:outputFormat>
<hr/>
```

6. Add a `DataTable` component to the `list.xhtml` file.
   a. Remove the Back Home `commandLink` placeholder:

      ```
      <h:commandLink action="home">Back Home</h:commandLink>
      ```

   b. Replace it with the `h:dataTable` tag. The value attribute should refer to the `DVDCollection` property from the `dvd` managed bean. The `var` attribute uses the variable item to reference each `DVDItem` element in the list.
   c. Use a separate column to render each field in `DVDItem`: `title`, `releaseyear`, and `genre`.
   d. Use a facet component to place a header at the top of each column.
   e. Use some additional table attributes to format the table.

   Following is an example of the table:

```
<h:dataTable value="#{dvd.DVDCollection}" var="item"
styleClass="table" headerClass="headers"
rowClasses="oddRows,evenRows" border="1" cellspacing="0"
cellpadding="5" frame="box">
   <h:column>
      <f:facet name="header">#{msg.column_Title}</f:facet>
      #{item.title}
   </h:column>
   <h:column>
      <f:facet name="header">#{msg.column_Year}</f:facet>
      #{item.releaseyear}
   </h:column>
   <h:column>
      <f:facet name="header">#{msg.column_Genre}</f:facet>
      #{item.genre}
   </h:column>
</h:dataTable>
```

7. Test by running the application.

   Navigate to the `list.xhtml` page to see the list of DVDs from the database.

# Practice 9-2: Enhancing the List: Adding a Scroll Bar

## Overview

In this practice, you improve the functionality of the data table by adding a simple scroll bar to the component.

## Assumptions

NetBeans is running.

## Tasks

1. Open the `list.xhtml` page.
2. Add a `<div>` tag element above the `<h:form>` tag. The `<div>` tag defines a section or division.
   a. Add the following tag:

      ```
      <div style="overflow: auto; width: 100%; height: 400px;">
      ```

   b. Close the tag appropriately.
3. Save the `list.xhtml` and retest the application. The `list` page should now have a scroll bar that fits inside the content area defined.

# Practice 9-3: Enhancing the List: Adding Sort

## Overview

In this practice, you add functionality to the data table by enabling the user to select the column title and sort the data table contents.

## Assumptions

NetBeans is running.

## Tasks

1. Create a new Java Package in the Source Packages folder in the `DVDLibrary` project called `com.example.util`.
   a. Copy the file D:\resources\sort\ItemComparator.java to the new package.
   b. Open the `ItemComparator.java` class and review the code. This is a utility class that compares two `DVDItem` objects and returns -1 if the two DVD items are equal (ignoring case) or 0 if they are not equal.
2. Copy the other two Java classes in the `D:\resources\sort` directory, `SortFilterModel.java` and `TableData.java` to the package: `com.example.beans`
   a. Open the `SortFilterModel.java` class and review the code. This class extends the abstract class `DataModel` and overrides some of the methods, enabling you to "wrap" the data.
   b. Open the `TableData.java` class and review the code. This class "intercepts" calls to the original data model, using an instance of the wrapper class `SortFilterModel` to provide methods to sort the rows in the wrapped `DataModel` by title, year, or genre.
3. Edit `DVDLibraryBean.java`.

   Note that `TableData` is looking for a `boolean` getter method `isDVDCollectionUpdated`.

   a. To fix this, you will need to add a property to `DVDLibraryBean` and a getter and setter method:

      ```
      private boolean DVDCollectionUpdated;
      ```

   b. Then change the `addDVD` method to set this `boolean` value to true if the `addDVD` method returns successfully and force a re-fetch of the records from the data store:

      1) Delete the following code:

      ```
      itembean.addItem(item);
      ```

      2) Add the following code:

      ```
      if( itembean.addItem(item)){
         setDVDCollectionUpdated (true);
         this.dvdCollection = null;
        }
      ```

4. Edit `ItemEJB.java`
   a. Modify `addItem` method.

```
   public boolean addItem(Item item) throws ItemException {
       boolean success = true;
        try {
             System.out.println(" item details"+item.getId());
            em.persist(item);
            em.flush();
            System.out.println(" item details"+item.getGenre());
        } catch (EntityExistsException pe) {
            success=false;
            throw new ItemException("Item with id " +
item.getId() + " exists.");
        }
        return success;
    }
```

5. Open the `list.xhtml` page.
   a. The `h:dataTable` now needs to come from `TableData`, rather than `DVDLibraryBean`, so replace the value element:

```
<h:dataTable value="#{tableData.library}" ...
```

   b. Replace the existing facet to add an action `commandLink` to the column title so that when clicked, each link will call the appropriate sort method in the `TableData` model. For example, for the title column:

```
<f:facet name="header">
   <h:commandLink action="#{tableData.sortByTitle}">
      #{msg.column_Title}
   </h:commandLink>
</f:facet>
#{item.title}
```

   c. Make similar changes for the year and genre column titles, by using `tableData.sortByYear` and `tableDate.sortByGenre` as action methods.
   d. Save the `list.xhtml` page.
6. Run the `DVDLibrary` project, log in, and try clicking the column titles to sort the rows by `title, releaseyear,` and `genre.`

# Practices for Lesson 10: Handling Events

**Chapter 10**

# Practices for Lesson 10: Overview

## Practices Overview

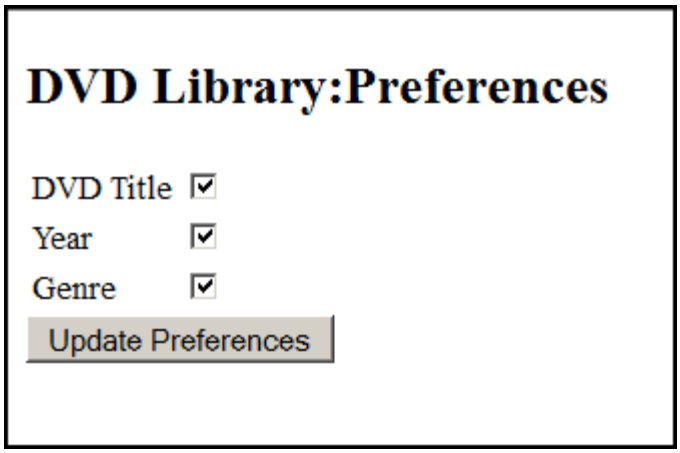
In these practices, you will implement the preferences page and add the capability of changing the language without having to alter the browser's language settings.

# Practice 10-1: Implementing the Preferences Page

## Overview

In this practice, you add a preferences page to the `DVDLibrary` application. This page uses a set of check boxes to select which columns are displayed on the `list.xhtml` page. You will use event handlers and listeners to implement the preferences page for the `DVDLibrary` project.

The preferences page will look as shown in the following figure:




## Assumptions

NetBeans is running and you have completed the previous practices.

## Tasks

1. Create a Java class with the following information:

| | |
|---|---|
| Class Name | `Prefs` |
| Package | `com.example.beans` |

   a. Add the annotations to scope this class by using session scope:

```
@Named
@SessionScoped
```

2. Add three `boolean` properties to the `prefs` bean, along with getters and setters:

```
private boolean title;
private boolean releaseyear;
private boolean genre;
```

**Note:** Make sure that you set the value of the three properties to true in the constructor, similar to the following; otherwise, the list will not show at all when you run the application.

```
public Prefs() {
   this.title = true;
   this.releaseyear = true;
   this.genre = true;
}
```

3. Open the `prefs.xhtml` page.

   a. Change the `commandLink` placeholder to a button that uses a message from the message bundle:

   ```
   <h:commandButton action="home" value="#{msg.btn_prefs}"/>
   ```

   b. Add a `panelGrid` component with two columns to the `form` component:

   ```
   <h:form>
      <h:panelGrid columns="2">
      </h:panelGrid>
   </h:form>
   ```

4. Lay out three text strings and three check box components in the panel grid. For the column title text, use the `column_Title`, `column_Year`, and `column_Genre` properties from the resource bundle `msg`. For the check box components, bind their value attributes to the `title`, `year`, and `genre` properties of the `prefs` managed bean:

```
<h:form>
   <h:panelGrid columns="2">
      #{msg.column_Title}
      <h:selectBooleanCheckbox id="title"
                               value="#{prefs.title}"/>
      #{msg.column_Year}
      <h:selectBooleanCheckbox id="year"
                               value="#{prefs.releaseyear}"/>
      #{msg.column_Genre}
      <h:selectBooleanCheckbox id="genre"
                               value="#{prefs.genre}"/>
   </h:panelGrid>
</h:form>
```

5. On the `list.xhtml` page, add a `rendered` attribute to each column element and bind the attribute value to the `title`, `releaseyear`, and `genre` properties of the `prefs` managed bean. For example, for the column that displays the title of the DVD, the column tag should be:

```
<h:column rendered="#{prefs.title}">
...
</h:column>
```

   a. Add the `rendered` attribute to each of the columns: `title`, `releaseyear`, and `genre`.

   b. Save the `list.xhtml` page.

6. Run the application and verify that you can use the `prefs` managed bean to control the columns of the DVD list table that you want to display.

   Note that you must click the Update Preferences button to retain the settings you choose on the `prefs` page. This is because the value of the changed check boxes is not set until the page is processed. You will change this behavior using AJAX in Practice titled “Using AJAX and Composite Components With JSF.”

# Practice 10-2: Implementing an Event Handler to Set the Locale

## Overview

In this practice, you add a feature to the `prefs` page to enable the user to switch languages programmatically by using an action listener.

The Preferences page will look as shown in the following figure:


## Assumptions

NetBeans is running and you have completed the previous practices.

## Tasks

1. Copy the `LocaleChanger.java` class from the `D:\labs\resources\locale` directory.
2. Paste the class to the `com.example.beans` package.
3. Open the `LocalChanger.java` class and review the code.

   Note the `langChanged` method:

   ```
   public void langChanged () {
      FacesContext context = FacesContext.getCurrentInstance();
      currLocale = new Locale(language);
      context.getViewRoot().setLocale(currLocale);
   }
   ```

   This method introduces the `FacesContext` object. `FacesContext` contains all the per-request state information related to the processing of a single JavaServer Faces request and the rendering of the corresponding response. It is passed to, and potentially modified by, each phase of the request processing life cycle. In this method, you are using the context and changing the locale of the context programmatically, specifying a different message bundle, and changing the application language without having to change the language of the browser.

4. Open the `prefs.xhtml` file.
   a. Add the JSF core library to the root `<html>` tag:

      ```
      xmlns:f="http://java.sun.com/jsf/core"
      ```

   b. Add the following code to change the locale above the `commandButton`:

      ```
      #{msg.lang}
      <h:panelGroup>
         <h:commandLink immediate="true"
                        action="#{locale.langChanged}">
            <f:setPropertyActionListener target="#{locale.language}"
                                         value="de"/>
            <h:graphicImage library="images" name="de_flag.gif"
                            style="border: 0px; margin-right: 1em;"/>
         </h:commandLink>
         <h:commandLink immediate="true"
                        action="#{locale.langChanged}">
            <f:setPropertyActionListener target="#{locale.language}"
                                         value="en"/>
            <h:graphicImage library="images" name="us_flag.gif"
                            style="border: 0px"/>
         </h:commandLink>
      </h:panelGroup>
      <p/>
      ```

   c. Save the `prefs.xhtml` file.

5. Copy the two images needed: `de_flag.gif` and `us_flag.gif`, from the `D:\labs\resources\images` directory to the `Web Pages/resources/images` directory in the `DVDLibrary` project.

6. Because the context is reset during the life cycle of the JSF application, you need to add a root view tag to the `masterLayout` template to use the language set by the managed bean `locale`.
   a. Open the `masterLayout.xhtml` file in the templates directory of the `DVDLibrary` project.
   b. Add the JSF core library to the root `<html>` tag:

      ```
      xmlns:f="http://java.sun.com/jsf/core"
      ```

   c. Add the root `view` tag as a top-level tag (just below the `<html>` tag):

      ```
      <f:view locale="#{locale.currLocale}">
      ```

   d. Close the `view` tag just above the `html` tag:

      ```
      ...
      </f:view>
      </html>
      ```

   e. Save the `masterLayout.xhtml` file.

7. Complete the following steps to add the `masterLayout` template to `prefs.xhtml`:
   a. Open `prefs.xhtml` and add the JSF facelets library to the root `<html>` tag:
      `xmlns:ui="http://java.sun.com/jsf/facelets"`
   b. Add the following tags above the `<form>` tag at the beginning of the page:

```
<ui:composition template="/templates/masterLayout.xhtml">

      <ui:define name="heading">

     #{msg.appName}:#{msg.title_prefs}



      </ui:define>
         <ui:define name="content">
    <div style="overflow: auto; width: 100%; height: 400px;">
```

   c. At the end of the page, add the following tags:

```
</div>
        </ui:define>
     </ui:composition>
```

8. Run the application and check the following:
   a. `prefs.xhtml` applies the `masterLayout` template
   b. The language changes dynamically when the appropriate flag is clicked on the `prefs` page.

# Practices for Lesson 11: Using AJAX and Composite Components with JSF

**Chapter 11**

# Practices for Lesson 11: Overview

## Practices Overview

In these practices, you will create composite components for the login page and use Ajax to immediately process changes on fields for the `add` and `prefs` pages.

# Practice 11-1: Creating a Composite Component

## Overview

In this practice, you create a composite component that encapsulates the components in the login form and use it in the `DVDLibrary` project.

## Assumptions

NetBeans is running and you have completed the previous practice.

## Tasks

1. Create a directory named `util` in the `Web Pages\resources` directory of the `DVDLibrary` project.
2. Create a JSF page named `signin.xhtml` in the newly created `util` directory.

   The file name `signin` becomes the composite component name.
3. Add the following composite namespace declaration to `signin.xhtml` to make the composite component library available:

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

4. Modify the `head` element to include the `styles.css` style sheet so that you can use the `error` styleClass that you defined earlier.

```
<h:head>
   <title>This is not displayed</title>
   <h:outputStylesheet library="css" name="styles.css" />
</h:head>
```

5. In the body element, declare the interface for the `signin` component:

```
<composite:interface>
   <composite:attribute name="usernamePrompt"/>
   <composite:attribute name="username" required="true"
                        type="String"/>
   <composite:attribute name="passwordPrompt"/>
   <composite:editableValueHolder name="password"/>
   <composite:attribute name="loginButtonText"/>
   <composite:attribute name="loginAction"
               method-signature="java.lang.String action()"/>
</composite:interface>
```

   The declaration specifies that the `signin` component expects attributes for the username prompt and username, for the password prompt and password, and for the login button text and action method.

6. Using the `panelGrid` component in the `login.xhtml` page as a reference, provide the implementation to the `signin` component.
   a. Add a two-column `panelGrid` component.
   b. Lay out the `username` and `password` components, and bind their value attributes to the composite component attributes, `username` and `password`, respectively:

```
<composite:implementation>
   <h:panelGrid columns="2">
       #{cc.attrs.usernamePrompt}
       <h:inputText id="username" value="#{cc.attrs.username}"
                    style="background-color: yellow"/>
       #{cc.attrs.passwordPrompt}
       <h:inputSecret id="password"
                      value="#{cc.attrs.password}"
                      style="background-color: yellow"
                      required="true" />
       <h:message for="password" showSummary="true"
                  showDetail="false" styleClass="error" />
       <h:commandButton id="loginButton"
                        value="#{cc.attrs.loginButtonText}"
                        action="#{cc.attrs.loginAction}"/>
   </h:panelGrid>
</composite:implementation>
```

   Note the use of the "`cc.attrs.`" notation. This enables the implementation to take "arguments" from the use of the composite, making it very flexible.
7. Save the composite component `signin.xhtml`.

## Practice 11-2: Using the Composite Component

### Overview

In this practice, you update the login page so that it uses the `signin` composite component.

### Assumptions

NetBeans is running and you have completed the previous practice.

### Tasks

1. Open the `login.xhtml` page.
2. Add the following namespace declaration to be able to access the composite component.

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

3. Modify the form element to use the `signin` component, providing values for each of the attributes defined in the composite implementation:

```
<h:form>
   <util:signin usernamePrompt="#{msg.usernamePrompt}"
                username="#{login.username}"
                passwordPrompt="#{msg.passwordPrompt}"
                password="#{login.password}"
                loginButtonText="#{msg.loginBtnTxt}"
                loginAction="#{dvd.login}"/>
</h:form>
```

   `loginAction` takes a method as its attribute.

4. You will need to add login method to the `DVDLibraryBean` using the `method-signature` that you defined in the composite interface, complete the following step:

   Browse to `src packages > com.example.beans> DVDLibraryBean.java.`

```
public String login () {
   return "home";
}
```

5. Run the `DVDLibrary` application.
6. Verify that the composite component is used to render the username and password fields, and the application works as before.

# Practice 11-3: Using AJAX

## Overview

In this practice, you use the built-in support for AJAX in JSF in the `DVDLibrary` project.

## Assumptions

NetBeans is running and you have completed the previous practice.

## Tasks

1. Open the `add.xhtml` page.
   a. The year field is currently evaluated only when the user attempts to add a DVD title. A more user-friendly approach would be to validate the year field as soon as the user enters the value. An easy way to do this (without having to create a listener method) would be to use the `ajax` tag:

```
<h:inputText id="year" value="#{dvd.year}" required="true">
   <f:convertNumber integerOnly="true" groupingUsed="false"/>
   <f:validateLongRange minimum="1909"
                        maximum="#{dvd.currentYear}"/>
   <f:ajax event="valueChange" render="yearError"/>
</h:inputText>
<h:message id="yearError" for="year" styleClass="error"
           showSummary="true" showDetail="false"/>
```

   The `f:ajax` tag calls the validators as soon as the web page detects a change in the year field (usually by clicking one of the other fields), and directs the results to the message with the ID "`yearError`".

   b. Save `add.xhtml`.
2. Run the `DVDLibrary` application and test to see that the year field reports an error as soon as it detects a change in the field.
3. Because AJAX processes immediately, you can improve the functionality of the preferences page by "Ajaxifying" the check boxes that you created on the `prefs` page:
   a. Open the `prefs.xhtml` page.
   b. Change the check box for the title to include an `ajax` call:

```
<h:selectBooleanCheckbox id="title"
                         value="#{prefs.title}">
   <f:ajax event="valueChange" execute="@this"/>
</h:selectBooleanCheckbox>
```

   This will force the check box to process the change as soon as the check box is selected or deselected.

   c. Make the same change to the `year` and `genre` check boxes.
   d. Remove `commandButton` (if you want) because it is no longer required to process changes to the preferences.
   e. Save the `prefs.xhtml` page.

4. Run the `DVDLibrary` application and test to see that the preferences are set immediately (without having to click the Update Preferences button).
   a. Click the **Set user preferences link** and check/un-check any item on the page.
   b. Now click the **Display the DVD Library link**. The table will be populated based on the selection made in the preferences page.

# Practices for Lesson 12: Developing Composite Components and Using AJAX

**Chapter 12**

# Practices for Lesson 12: Overview

## Practices Overview

In this practice, you will add a custom component, spinner, to DVDLibrary project.

# Practice 12-1: Using a Custom Component

## Overview

In this practice, you add a custom component, spinner to the `add.xhtml` page of the `DVDLibrary` project to enter the year instead of `<h:inputText>` while adding a new DVD.

## Assumptions

NetBeans is running and you have completed the previous practice.

## Tasks

1. Create a new package under `source packages` called `com.example.component`.
2. Copy `UISpinner.java` from the `D:\labs\resources` folder to the package.

   This file describes the spinner custom component.
3. Edit `add.xhtml`.
   a. Open `add.xhtml`.
   b. Import the custom component namespace `<xmlns:corejsf="http://corejsf.com">`
   c. Delete `<h:inputText>` element and the following code to include spinner component.

```
<corejsf:spinner value="#{dvd.releaseyear}" minimum="1909"
maximum="#{dvd.currentYear}" size="5"/>
```

   d. Save `add.xhtml`.
4. Edit `web.xml`.
   a. Open `web.xml`.

```
   Add the <context-param> element.
     <context-param>
        <param-name>javax.faces.FACELETS_LIBRARIES</param-name>
        <param-value>/WEB-INF/corejsf.taglib.xml</param-value>
    </context-param>
```

   b. Save `web.xml`.
5. Create a Tag Library Descriptor.

Create an `.xml` file, `corejsf.taglib.xml` and save it in the `WEB-INF` folder.

   a. Right-click the project, select `New> Other`.
   b. In the **Categories** column, select **XML** and in the **File Types** select **XML Document**.
   c. Click Next.

d. In the New XML Document window, provide the following details:

Name: `corejsf.taglib`

Folder: Click browse and select `WEB-INF` folder by expanding `web` folder.

e. Click Select Folder.

f. Click Next

g. Click Finish.

Edit the file by adding `<facelet-taglib>`

```
<facelet-taglib version="2.0"
   xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
   http://java.sun.com/xml/ns/javaee/web-
facelettaglibary_2_0.xsd">
   <namespace>http://corejsf.com</namespace>
   <tag>
      <tag-name>spinner</tag-name>
      <component>
         <component-type>com.corejsf.Spinner</component-type>
      </component>
   </tag>
</facelet-taglib>
```

6. Run the `DVDLibrary` application.

7. Verify that the while adding a new DVD, the spinner component enables you to select the year, and the application works as before.

# Practices for Lesson 13: HTML 5 with JSF 2.0

**Chapter 13**

# Practices for Lesson 13: Overview

## Practices Overview

In this practice, you will use a HTML 5 element `<progress>` in `DVDLibrary` project for the login functionality.

# Practice 13-1: Using a `<progress>` element

## Overview

In this practice, you will use a HTML 5 element `<progress>` in `DVDLibrary` project for the login functionality.

## Assumptions

NetBeans is running and you have completed the previous practice.

## Tasks

1. Open `login.xhtml`.
2. Delete the `<h:form>` element and add the below code.

```
<h:form >
 <h:panelGrid columns="2">
 #{msg.usernamePrompt}:
<h:inputText id="username" value="#{login.username}"/>
 <p/>
#{msg.passwordPrompt}:
<h:inputSecret id="password" value="#{login.password}"/>
 </h:panelGrid>
  <p/>
</h:form>
```

   Note that the Login button has been removed.

3. Make the following changes to the `<h:form>` elements:
   a. Add id to the `<h:form>` element.

```
<h:form id="LoginForm">
```

4. Add `<progress>` element to the form just after the `</h:panelGrid>`.

```
<p><progress id="bar" value="0" max="100"></progress>
```

   This creates a progress bar with minimum value of 0 and max value of 100.

5. Add a `<span>` element `wait` before the `<progress>` element to display the message "Please wait..."

```
<p><span id="wait"> </span></p>
```

6. Add the `<span>` and `<h3>` elements to display the status of the progress bar. These two elements are added just below the `<progress>` element in the form.

```
<span id="status"> </span>
 <h3 id="finalMessage"></h3></p>
```

7. Add a JavaScript method, `showProgress(0)` which controls the status of the progress bar. The JavaScipt is added after this tag `<ui:define name="content">`.

```
  <script type="text/javascript">

   var loaded=0;
  function showProgress(loaded)
     {

     var bar = document.getElementById('bar');
    var status = document.getElementById('status');

document.getElementById('wait').innerHTML="Please wait...";
                         status.innerHTML=loaded+"%";
                        bar.value=loaded;
                        loaded++;
                        var
sim=setTimeout("showProgress("+loaded+")",100);

  if(loaded==100)
     {

status.innerHTML="100%";

document.getElementById('status').innerHTML="Login
successful...";
     bar.value=100;
     clearTimeout(sim);

window.location.href="faces/index.xhtml";



                        }

                    }
                </script>
```

This method controls the status of the progress bar from zero to 100 and when the value reaches 100 it redirects to the `index.xhtml`.

8. The `showProgress(0)` method is invoked on `blur` event of the password field, `<h:inputSecret>`. To implement, you have to bind an `<f:ajax>` call on blur of the password field.

   Modify the `<h:inputSecret>` as below

```
 <h:inputSecret id="password" value="#{login.password}">
  <f:ajax event="blur"  onevent="showProgress(0)"
execute="username password" immediate="true" />
 </h:inputSecret>
```

9. Import `<xmlns:f=http://java.sun.com/jsf/core>` namespace.
10. Save `login.xhtml`.
11. Run the application. Verify the login feature-
    a. Enter username and password.
    b. Click near the `progress` element. The `progress` element value increases from 0 to 100.
    c. You are redirected to `index.xhtml`.

# Practices for Lesson 14: Configuring and Securing JSF Applications

**Chapter 14**

# Practices for Lesson 14: Overview

## Practices Overview

In these practices, you will configure declarative Java EE security.

# Practice 14-1: Using Declarative Security

## Overview

In this practice, you add users to an application server, map application roles to server users, and create all the elements for a login form.

## Assumptions

You have completed the practice for lesson 13.

The database is running.

## Tasks

1. Open the `DVDLibrary` project in NetBeans if it is not already open.
2. Using the Services tab in NetBeans, start your Oracle WebLogic server if it is not already running.
3. Create three user accounts using the WebLogic console.
   - Open the http://localhost:7001/console/ URL in a web browser and log in.
     - As a reminder, the username is `weblogic` and password is `welcome1`.
   - In the Domain Structure box click Security Realms.
   - In the Realms table click the `myrealm` realm.
   - Click the Users and Groups tab.
   - In the Users subtab, create three new users by clicking the New button.
     - User 1 Name: `duke`
     - User 1 Password: `welcome1`
     - User 2 Name: `andy`
     - User 2 Password: `welcome2`
     - User 3 Name: `pat`
     - User 3 Password: `welcome3`
4. Modify the `com.example.beans.LoginBean` CDI managed bean.
   - Add an `isAdmin` method to the `LoginBean` class.

```
public boolean isAdmin() {
    ExternalContext externalContext =
FacesContext.getCurrentInstance().getExternalContext();
    return externalContext.isUserInRole("admin");
}
```

- Add a `login` method to the `LoginBean` class, which uses the `HttpServletRequest.login` method.

```
public String login() {
    ExternalContext externalContext =
FacesContext.getCurrentInstance().getExternalContext();
    HttpServletRequest request = (HttpServletRequest)
externalContext.getRequest();
    try {
        request.login(userName, password);
        return "index";
    } catch (ServletException ex) {

Logger.getLogger(LoginBean.class.getName()).log(Level.INFO,
"Failed to log in {0}", userName);
        FacesContext facesContext =
FacesContext.getCurrentInstance();
        FacesMessage facesMessage = new FacesMessage("Failed to
log in");
        facesMessage.setSeverity(FacesMessage.SEVERITY_ERROR);
        facesContext.addMessage(null, facesMessage);
        return null;
    }
}
```

- Add a `logout` method to the `LoginBean` class.

```
public String logout() {
    ExternalContext externalContext =
FacesContext.getCurrentInstance().getExternalContext();
    HttpServletRequest request = (HttpServletRequest)
externalContext.getRequest();
    try {
        request.logout();
    } catch (ServletException ex) {

Logger.getLogger(LoginBean.class.getName()).log(Level.SEVERE,
"Failed to logout", ex);
    }
    return null;
}
```

5. Modify the `login.xhtml` Facelet page.
   - Remove the code that was added to use a progress bar. `login.xhtml` should contain:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:util="http://java.sun.com/jsf/composite/util">
    <head><title>IGNORED</title></head>
    <body>
        <ui:composition
template="/templates/masterLayout.xhtml">
            <ui:define name="heading">
                #{msg.appName}:#{msg.loginHeader}
            </ui:define>

            <ui:define name="sideBarLeft">
                <ui:include
src="/sections/login/sideBarLeft.xhtml"/>
            </ui:define>

            <ui:define name="content">
                <h:form rendered="#{request.userPrincipal eq
null}">
                    <util:signin
usernamePrompt="#{msg.usernamePrompt}"
                                 username="#{login.username}"

passwordPrompt="#{msg.passwordPrompt}"
                                 password="#{login.password}"

loginButtonText="#{msg.loginBtnTxt}"
                                 loginAction="#{login.login}" />
                </h:form>
                <h:form  rendered="#{request.userPrincipal ne
null}">
        You are logged in as #{request.userPrincipal.name}<br/>
        <h:commandButton value="#{msg.logoutBtnTxt}"
action="#{login.logout}" text="Logout"/>
    </h:form>
</ui:define>
        </ui:composition>
    </body>
</html>
```

6. Add a logout link to the `sideBarleft.xhtml` file located at: `Web Pages -> Sections -> dvdlibrary`
   - Open the `sidebarLeft.xhtml` page in the `Web Pages/Sections/dvdlibrary` directory.
   - Modify the last `<h:commandLink>` to the following snippet

```
<hr/>
                <h:commandLink value="#{msg.logoutBtnTxt}"
action="#{login.logout}"/>
            </h:form>
```

7. Test the login form to verify if authentication has been applied.
   - Deploy the `DVDLibrary` application.
   - Open the application by visiting http://localhost:7001/DVDLibrary/ in a web browser.
   - The Login screen should show up. Log in as duke, andy, and pat. In other login names or incorrect passwords, you will not be authenticated and will not be allowed to see the home page of the application.

   **Note:** If you fail to enter the correct password five times in a row, the account will be locked for 30 minutes. You will see a message in the log file that says:

```
<Apr 2, 2013 3:04:18 AM CDT> <Notice> <Security> <BEA-090078>
<User duke in security realm myrealm has had 5 invalid login
attempts, locking account for 30 minutes.>
```

   You can reset the accounts (remove the lockout) by restarting WebLogic Server.

8. In this task, you will add authorization to the `DVDLibrary` application. You will create two roles, `user` and `admin`. The user role is authorized to only view the items but the admin role is authorized to view the items and also add items. Complete the following steps:
   a. Open web.xml. Add the following below the `</context-param>` tag, save and close the file.

```
<security-role>
        <description>user role to list items</description>
        <role-name>user</role-name>
    </security-role>
    <security-role>
        <description>admin role to add items. </description>
        <role-name>admin</role-name>
    </security-role>:
```

b. Open `weblogic.xml` Add the following below the `</context-root>` element, close and save the file:

```
<security-role-assignment>
    <role-name>admin</role-name>
    <principal-name>duke</principal-name>
    <principal-name>andy</principal-name>
  </security-role-assignment>

    <security-role-assignment>
      <role-name>user</role-name>
      <principal-name>pat</principal-name>
  </security-role-assignment>
```

c. Create a new `xhtml` page in the Web Pages node and name it as `notAuthorized.`xhtml. The file should contain the following:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!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"
      xmlns:ui="http://java.sun.com/jsf/facelets">
    <head>
        <title>IGNORED</title>
    </head>
    <body>
        <ui:composition
template="/templates/masterLayout.xhtml">
            <ui:define name="heading">
                #{msg.appName}:#{msg.title_message}
            </ui:define>

            <ui:define name="content">

                <div> You are not authorized to perform this
task </div>
            </ui:define>
        </ui:composition>
    </body>
</html>
```

d. Open the `faces-confix.`xml file located at `Web Pages -> WEB-INf node.` Modify the navigation rule by adding a navigation case for the `add` option in the file as follows and then close and save the file.

```
<navigation-rule>
        <from-view-id>/*</from-view-id>
        <navigation-case>
            <from-outcome>add</from-outcome>
            <if>#{empty login.username}</if>
            <to-view-id>/login.xhtml</to-view-id>
        </navigation-case>
        <navigation-case>
            <from-outcome>add</from-outcome>
            <if>#{login.admin eq false }</if>
            <to-view-id>/notAuthorized.xhtml</to-view-id>
        </navigation-case>
    </navigation-rule>
```

9. As an alternative to **step d** in the previous task, you can also modify the `add` method of the CDI bean to execute only if the logged in user belongs to the `admin` role. Complete the following steps:

   a. Open the `DVDLibraryBean.java` from the `com.example.beans` package. Locate the `addDVD` method.

   b. Add an `if else` statement to the method as shown in bold in the following code snippet:

```
public String addDVD() throws PreexistingEntityException,
Exception {
  if(login.isAdmin())
        {
        if(!newGenre.equals("")) {
            addGenre(newGenre);
            setGenre(newGenre);
        }
------------
releaseyear = getCurrentYear();
   setNewGenre("");
        }
        else{
            System.out.println("not allowed");
            FacesContext facesContext =
FacesContext.getCurrentInstance();
        FacesMessage facesMessage = new FacesMessage("Addition
Not allowed, you are not an admin");
```

```
        facesMessage.setSeverity(FacesMessage.SEVERITY_ERROR);
        facesContext.addMessage(null, facesMessage);
        return null;
        }
  return "home";
   }
```

10. Test the `DVDLibrary` application.
    - Deploy the `DVDLibrary` application.
    - Open the application by visiting http://localhost:7001/`DVDLibrary`/ in a web browser.
    - Use the Login screen to log in with the `pat` account.
    - You can view the list of DVD items.
    - Attempt to click the Add option on the side bar. As `pat`, you are not in the admin role and should not be able to add any item. You will get the `not allowed` message.
    - Adding an item should succeed while logged in as `duke` or `andy` that belongs to the admin role. You can log out and try to add an item while logged in as `duke` or `andy`.

# Practices for Lesson 15: Using Third Party Library for JSF Development

**Chapter 15**

# Practices for Lesson 15: Overview

## Practices Overview

In these practices, you use the Rich Faces component library, use the Trinidad component library, and develop a mobile web interface for an application.

# Practice 15-1: Using Prime Faces Component Library

## Overview

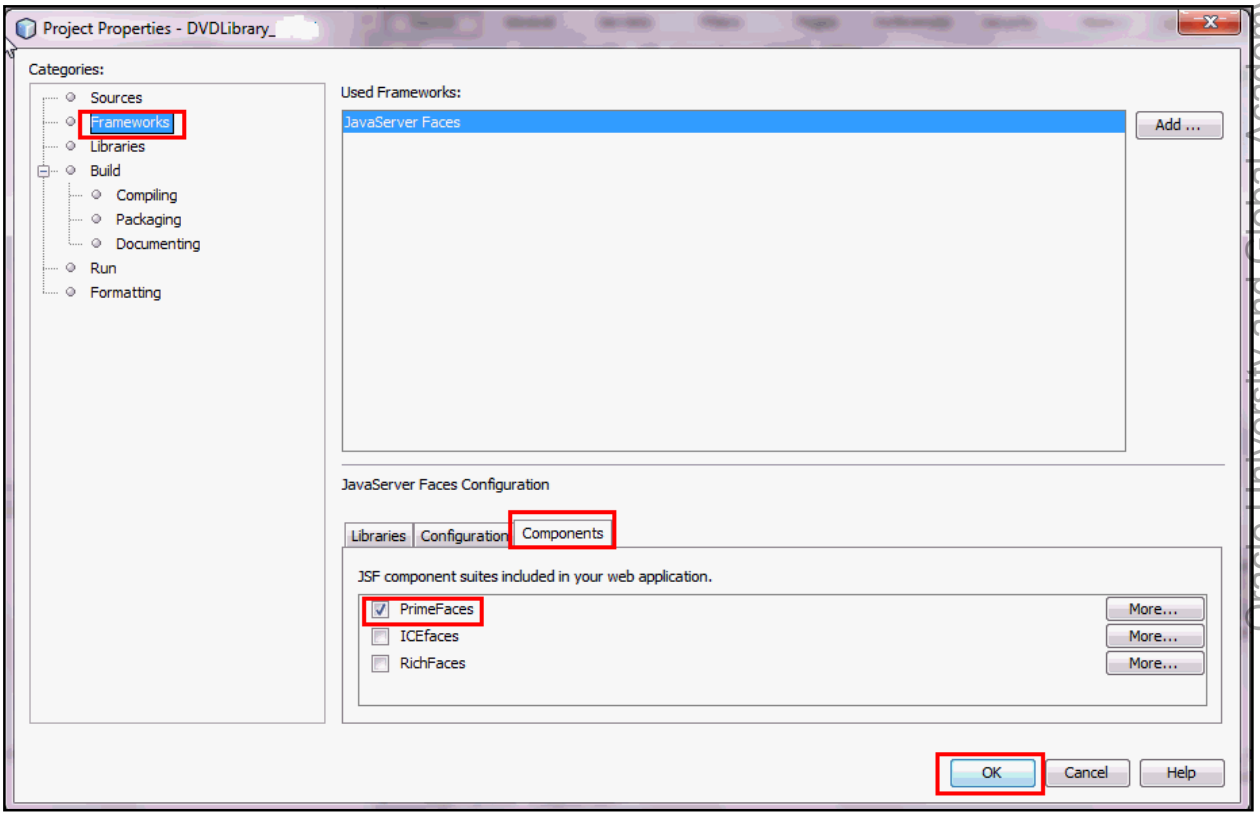
In this practice, you enable the `DVDLibrary` application to display pie chart to show the distribution of movies across the various genres. You will use the Prime Faces component library to include a chart in DVDLibrary application.

## Assumptions

You have completed the previous practice.

## Tasks

1. Open the `DVDLibrary` project in NetBeans if it is not already open.
2. Using the Services tab in NetBeans, start your Oracle WebLogic server if it is not already running.
3. Add the Prime Faces component library to the project. Complete the following steps:
   a. In the Projects window, right click `DVDLibrary` and select Properties.
   b. In the Project Properties dialog box, select **Frameworks** from the **Categories** section
   c. Select '**Prime Faces**' from the Components tab and click OK as shown in the following screenshot.



4. Create a CDI bean, for tha Chart properties. Complete the following steps:
   a. Create a Java class in the `com.example.beans` package and name it as `GenreChartBean`.
   b. Add the `@Named("chartBean")` and `@SessionScoped` annotation.
   c. Declare a `PieChartModel` component.
   d. Create a `CreatePieChartModel` method that will construct the pie chart based on the data available. The class should look as following:

```
@Named("chartBean")
@SessionScoped
public class GenreChartBean implements Serializable {
   private PieChartModel pieModel;

    public GenreChartBean() {
        createPieModel();
    }

    public PieChartModel getPieModel() {
        return pieModel;
    }
    private void createPieModel() {
        pieModel = new PieChartModel();

        pieModel.set("Action", 2);
        pieModel.set("Comedy",1);
        pieModel.set("Mystery", 3);
        pieModel.set("Drama", 6);
        pieModel.set("Sci Fi",2);
    }
}
```

**Notes:**

– You can refer to: http://www.primefaces.org/showcase-labs/ui/home.jsf to obtain the codes for several other components.
– In this example, the genre name and the count has been hard-coded.

5. Create a new facelet in Web Pages node.
   a. In the Projects window, expand `DVDLibary > Web Pages`. Right Click `Web Pages > New > XHTML..` Nme the file `Chart.xhtml`

b. Ensure that the file follows the same template as the rest of the pages. Add the namespace for the `Primae Faces tags`. The `<html>` tag should have the following namespace declarations:

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

c. Add the `<p:piechart>` tag to insert a pie chart component within the content `<ui:define>` tag. Using EL, assign the `value` attribute to `chartBean.pieModel`.

d. `Chart.xhtml` should look as follows:

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

   <head><title>IGNORED</title></head>
<body>
   <ui:composition template="/templates/masterLayout.xhtml">
      <ui:define name="heading">
        <h:outputFormat value="#{msg.welcome_list}">
   <f:param value="#{dvd.size}"/>
</h:outputFormat>
<hr/>
      </ui:define>
       <ui:define name="content">
    <div style="overflow: auto; width: 100%; height: 400px;">
  <h:form>

<p:pieChart id="custom" value="#{chartBean.pieModel}"
legendPosition="e" fill="true" showDataLabels="true"
                title="Genre Distribution"
style="width:400px;height:300px" sliceMargin="5" diameter="150"
/>

      </h:form>
</div>
</ui:define>
     </ui:composition>
    </body>
</html>
```

6. Add a link to view the chart in the left bar of the pagein the following way:

   In the Projects window, go to `DVDLibrary > Web Pages > sections >dvdlibrary`. Open `sideBarLeft.xhtml`. Add a `commandLink` to view the genre chart as shown in the following code snippet:

```
<h:form>
                <hr/>
                <h:commandLink value="#{msg.goToList_library}"
action="list"/><p/>
                <h:commandLink value="#{msg.goToAdd_dvd}"
action="add"/><p/>
                <h:commandLink value="#{msg.goToSet_prefs}"
action="prefs"/><p/>
                <h:commandLink value="#{msg.goToView_chart}"
action="Chart"/>
                <hr/>
                <h:commandLink value="#{msg.logoutBtnTxt}"
action="#{login.logout}"/>
            </h:form>
```

7. Modify the message properties files for both the languages. Complete the following steps:

   a. In the Projects window, expand `DVDLibrary > Source Pacakges > com.dvdlibrary.resources`

   b. Click `messages.properties` to open it and add the following name value pair:

```
goToView_chart=View genre chart
```

   c. Repeat the previous step for the `message.de.properties` file:

```
goToView_chart=Sehen genre diagramm
```

8. Deploy the `DVDLibrary` project, run it, and test the 'View Chart' option.

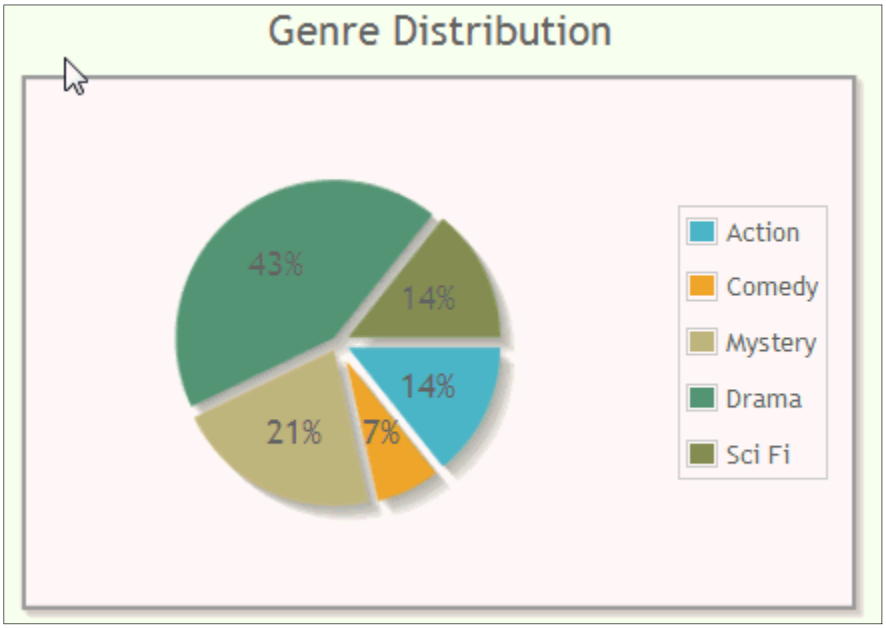
   a. Log in using the following credentials:

      User Name: `duke`

      Password: `welcome1`

   b. At the home page, from the side bar, select 'View genre chart'

c. Validate if you can see the pie chart depicting the distribution of DVDs in various genres in percentage as shown below:

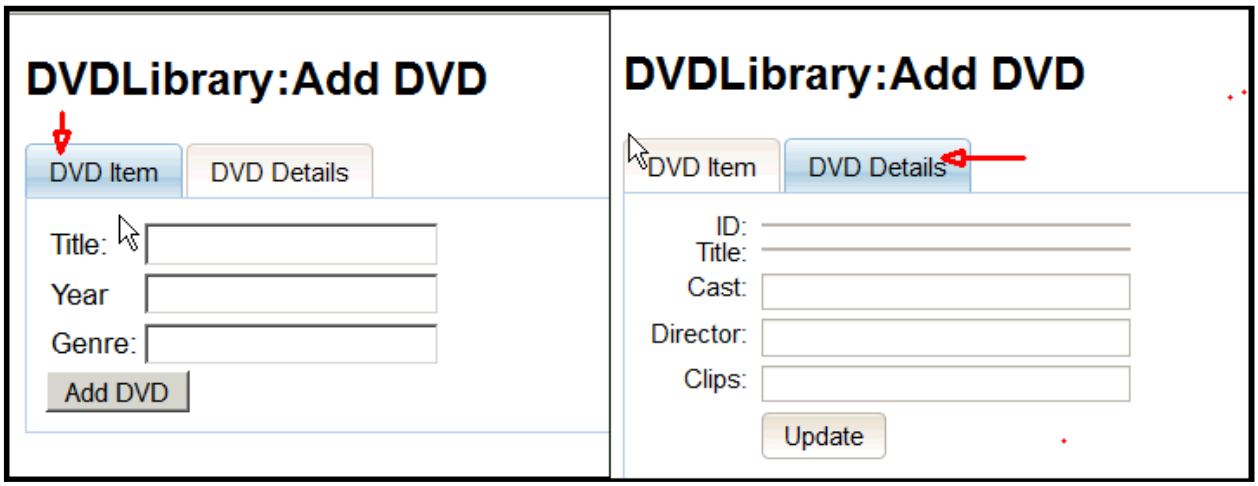


## Practice 15-2: Using the Trinidad Component Library

### Overview

In this practice, you use the Trinidad component library. The `add.xhtml` page will now contain a View and a Details tab. The View tab allows you to add DVD information as before. The Details tab displays a form, which allows further details of the DVD item to be updated.

The `add.xhtml` page will be displayed with two tabs as shown in the following images:




### Assumptions

NetBeans is running.

### Tasks

1. Define the required Trinidad libraries within NetBeans.
   a. Click the Tools menu in NetBeans.
   b. Click the Ant Libraries menu item.
   c. In the Ant Library Manager Dialog box, click the New Library button.
   d. Enter a library name of `Trinidad API 2.0.1` and click OK.
   e. With the `Trinidad API 2.0.1` library selected, click the Add JAR/Folder button.
   f. Select the `D:\labs\resources\trinidad-assembly-2.0.1\lib\trinidad-api-2.0.1.jar` file and click the Add JAR/Folder button.
   g. Repeat this process to add a library named `Trinidad Impl 2.0.1`, which contains the `D:\labs\resources\trinidad-assembly-2.0.1\lib\trinidad-impl-2.0.1.jar` file.

2. Open the required `DVDLibrary` project. Complete the following steps:
   a. Close the `DVDLibrary` project that is open in NetBeans.
   b. In the Services window, expand the `Servers > Oracle Web Logic Server >Applications >Web Applications` node. Select `DVDLibary`, right-click and select **undeploy.**
   c. Open the `DVDLibrary` project located at `D:\labs\15_ThirdParty\practices` folder.
3. Add the newly defined Trinidad libraries to the `DVDLibrary` project.
   a. Right-click the `DVDLibrary` and select Properties.
   b. In the Project Properties dialog box, select the Libraries category.
   c. Click the Add Library button, select `Trinidad API 2.0.1`, and click the Add Library button.
   d. In the Project Properties dialog box, select the Build -> Packaging category. Click the Add Library button, select `Trinidad Impl 2.0.1`, and click the Add Library button.
   e. Double-click the `Path in WAR` value, change it to `/WEB-INF/lib` and press the Enter key. Click the OK button. Note that failure to press the Enter button will result in the `Path in WAR` value not being saved.
4. Update the `web.xml` file to configure Trinidad parameters and URLs.
   a. Open the Favorites window. Enable using the Window menu if needed.
   b. Right-click in the Favorites menu and add the `D:\labs\resources` directory if it is not already present.
   c. Open the `D:\labs\resources\web.xml` file. Select all the lines within the file (Ctrl + A) and copy them (CTRL + C).
   d. In the Projects window, open the `web.xml` file from the `DVDLibrary` project. Paste the lines copied in the previous task, before the `<context-param>` tag. Remove any duplicate `<context-param>` tags.
5. Update the `faces-config.xml` file in the `WEB-INF` directory, to specify the render-kit.
   a. Populate the file as follows:

```
    <application>
------
    <!-- Use the Trinidad RenderKit -->
        <default-render-kit-id>
            org.apache.myfaces.trinidad.core
        </default-render-kit-id>
    </application>
```

6. Modify `DVDLibraryBean.java`.

   Add an `updateDVD` method as follows:

```
public String updateDVD(Item item) throws  Exception {

        //code to update the additional details of the DVD
         itembean.updateItem(item);
        return "index";
    }
```

7. Modify the `add.xhtml` page.

   a. A Trinidad Facelet page begins differently than a typical JSF Facelet. Replace the beginning of `add.xhtml` (up to the opening `<h:body>` tag, including `<h:body>`) with the following:

```
<tr:document xmlns="http://www.w3.org/1999/xhtml"
             xmlns:h="http://java.sun.com/jsf/html"
             xmlns:ui="http://java.sun.com/jsf/facelets"
             xmlns:f="http://java.sun.com/jsf/core"
             xmlns:tr="http://myfaces.apache.org/trinidad"
          xmlns:util="http://java.sun.com/jsf/composite/comps"
             title="DVD Manager">
```

   b. Replace the closing `</h:body>` and `</html>` tags with a single line:

```
</tr:document>.
```

   c. Add a Trinidad tabbed panel with two tabs, DVD Item and DVD Details, to the `add.xhtml` page. The View tab should contain your existing panel grid and commandlink tags to add a DVD.

```
<tr:form>
    <tr:panelTabbed position="above">
        <tr:showDetailItem text="Add DVD Item">
```

   **Note:** Existing `panelGrid` and CommandLink tags here.

```
        </tr:showDetailItem>
        <tr:showDetailItem text="DVD Details">


        </tr:showDetailItem>
    </tr:panelTabbed>
</tr:form>
```

   d. Using the web browser, visit http://localhost:7001/DVDLibrary/. You should now see a DVD Item tab and a DVD Details tab. Try switching back and forth between them.

e. Create a form inside the Details tab in the `add.xhtml` file. Forms cannot normally be nested inside of another form. Trinidad has a special `subform` tag that allows form nesting. You will also use Trinidad form controls that combine labels, input, and error messaging into a single tag.

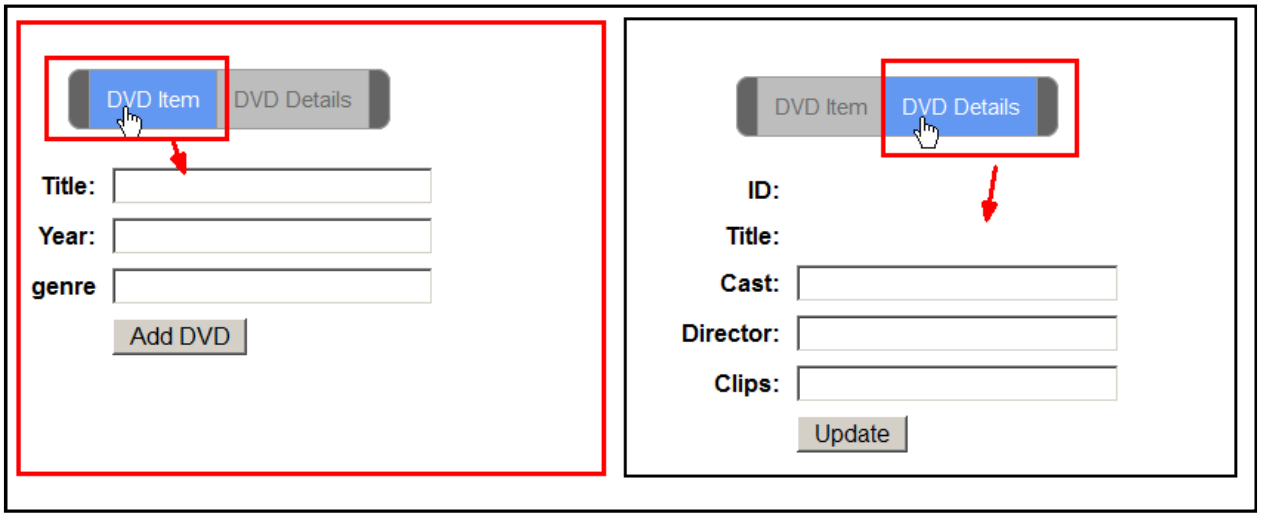
```
<tr:subform>
<tr:panelFormLayout>
<tr:inputText columns="25" label="ID:" readOnly="true" />
<tr:inputText columns="25" label="Title:" readOnly="true"  />
<tr:inputText columns="25" label="Cast:" />
<tr:inputText columns="25" label="Director:" />
<tr:inputText columns="25" label="Clips" />
<tr:commandButton action="#{dvd.updateDVD}" text="Update"/>
</tr:panelFormLayout>
</tr:subform>
```

f. Deploy the DVDLibrary application and view the two tabs with forms.

# Practice 15-3: Creating Mobile Web Applications (Optional)

## Overview

In this practice, you use Trinidad to develop a mobile phone interface to the Add DVD page of the `DVDLibrary` application. This is primarily achieved through dynamically selecting a Trinidad skin or style sheet based on browser type. The following are two screenshots of this practice when viewed in a mobile browser environment.



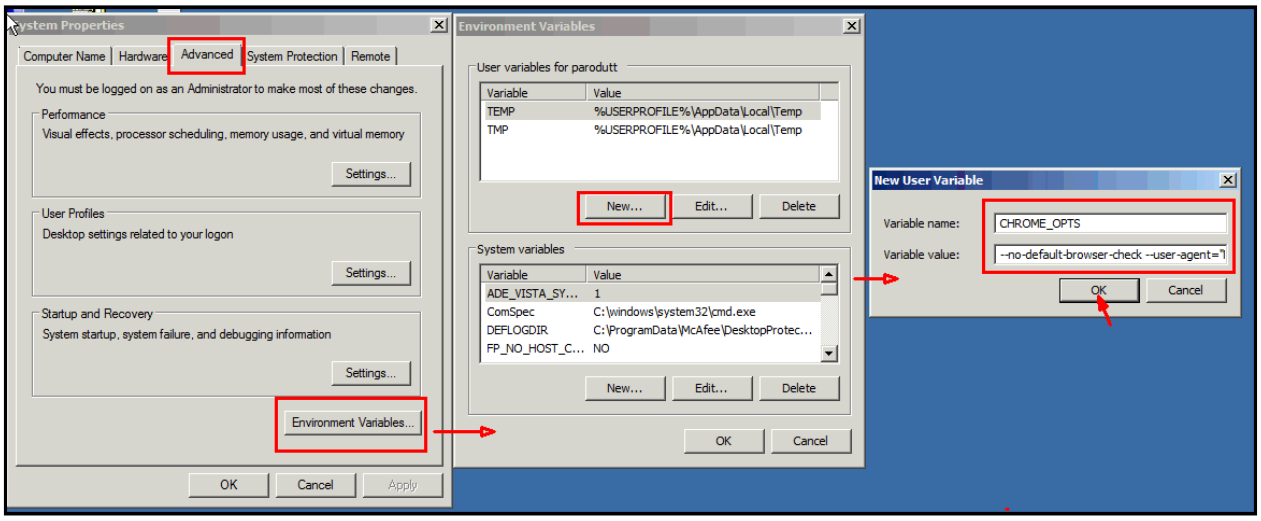

## Assumptions

You have completed the previous practice.

## Tasks

1. View the `DVDLibary` in a simulated mobile interface.
   a. Deploy the `DVDLibary` application.
   b. Simulate a mobile web browser environment. Complete the following steps:
      1) Create a copy of the Chrome shortcut on your desktop and name it "Mobile Chrome".
      2) Right-click the "Mobile Chrome" shortcut and click Properties.
      3) In the Properties window go to the **Target** text box. Go to the end of the text and add a space after the closing quotes. Enter `%CHROME_OPTS%` after the space
      4) Click OK.
      5) Right click **Computer** icon on your desktop and select **Properties** option.
      6) Create a new environmental variable and name it as `CHROME_OPTS`.

The following image shows the sequence of screens you get when you create a new environment variable:




7) Enter the value of the new variable as follows:

```
--no-default-browser-check --user-agent="Mozilla/5.0 (iPhone;
U; CPU iPhone OS 4_3_3 like Mac OS X; en-us)
AppleWebKit/533.17.9 (KHTML, like Gecko) Version/5.0.2
Mobile/8J2 Safari/6533.18.5"
file:///D:/labs/resources/mobilelauncher/mobileloader.html
```

c. Start Chrome with the Mobile Chrome shortcut located on the desktop.

d. The Mobile Chrome shortcut will launch Chrome using a `--user-agent` option with a value that mimics an iPhone. Note that the `-user-agent` option functions only if no other Chrome windows are open.

e. The Mobile Chrome shortcut will open the `D:\labs\resources\mobilelauncher\mobileloader.html` file, which will help you to resize your browser window to the approximate screen size of a mobile device. Resize the window until the viewport is around 320x480 pixels in size.

f. Open the http://localhost:7001/DVDLibrary/ URL in the Mobile Chrome window.

g. Note that the look of the application has not changed yet. This is approximately what a mobile phone user would see when viewing the `DVDLibrary` application.

2. Copy resource files to the `DVDLibrary` project.

a. The mobile interface and the desktop interface use the `modernDesktop.css` and `mobileWebKit.css` files, respectively. Copy the two files to the `DVDLibrary >Web Pages > css` directory from `D:\labs\resources\css`.

b. Copy the `trinidad-config.xml` and `trinidad-skins.xml` from `D:\labs\resources` to the `WEB-INF` directory of the `DVDLibrary` project.

c. Inspect the `trinidad-config.xml` file. Notice that it uses the `UserAgentBean` (a request-scoped bean) to dynamically determine the name of the Trinidad skin that should be used.

d. Inspect the `trinidad-skins.xml` file. Notice that the skin-family specified by the EL statement in the `trinidad-config.xml` file is connected to a CSS file here.

3. Modify the `add.xhtml` page to use the Trinidad `document` tag, which will be responsible for returning the correct CSS response to the end user.
   a. Replace the text from the beginning of the `add.xhtml` file up to and including the `<h:body>` with:

```
<?xml version='1.0' encoding='UTF-8' ?>
<tr:document xmlns="http://www.w3.org/1999/xhtml"
             xmlns:h="http://java.sun.com/jsf/html"
             xmlns:ui="http://java.sun.com/jsf/facelets"
          xmlns:util="http://java.sun.com/jsf/composite/comps"
             xmlns:f="http://java.sun.com/jsf/core"
             xmlns:tr="http://myfaces.apache.org/trinidad"
             title="DVD Manager">
```

   b. Replace the `</h:body>` and `</html>` tags at the end of the file with:

```
</tr:document>
```

4. In add`.xhtml` file, insert the following lines as the first element inside the `<tr:document>` tag (around line 9 or 10):

```
<f:facet name="metaContainer">
    <meta name="viewport" content="user-scalable=no,
width=device-width" />
</f:facet>
```

   **Note:** The viewport statement tells mobile browsers that the website fits within the viewable area of the browsers (no need to zoom in and out). This statement will not affect the behavior of a desktop web browser.

5. Test the mobile and desktop versions of the Add DVD page of the `DVDLibrary` application.
   a. Using the Mobile Chrome web browser shortcut, visit http://localhost:7001/DVDLibrary /
   b. Click the **Login** link and enter `Duke` and `welcome1` as the credentials.
   c. Click the **Add a DVD to my Collection** link and view the `add.xhtml` page
      – Verify the change in the skin and the display size of the two tabs of the page.

   **Note:** If you click the Add DVD button, it will not add a DVD to the Item table as the text boxes in the Add form have not been associated with the fields of the `DVDLibaryBean`. The objective of this practice is to view the application in a mobile environment and verify the look and feel of the add page.