SCHOOL OF COMPUTER SCIENCE

PRIFYSGOL
BANGOR
UNIVERSITY

Java Technologies Exercises - Notes

Java Server Faces

Dave Perkins

Introduction

In this laboratory session we investigate another Java web technology known as *Java Server Faces* (JSF). The JSF technology provides a framework for developing sophisticated web applications structured in accordance with the standard Three Tier Model outlined below:

- Presentation Tier (the web browser)
- Business Logic Tier (JSF container, JSF pages and the JavaBeans)
- Storage Tier (the DBMS)

Using JSF, developers present an application's content using facelets[1] , a mixture of XHTML and specialised JSF mark-up tags(listed in red). This approach is illustrated in the facelet below which generates a simple welcome message.

Note that JSF tags are marked in red.

```
<?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>Hello Facelet</title>
</h:head>

<h:body>
        Welcome to the world of JSF!
</h:body>
</html>
```

Amongst other things JSF technology supports

- Page layout
- Management of JavaBean components
- Placing *ready-made* user interface components onto web pages
- Defining navigation rules from one page to another
- Session tracking
- Data validation

Taken together these features ensure that the JSF API can provide a standard framework for building presentation tiers for a wide variety of web applications.

[1] The term *facelet* is a, perhaps humorous, reference to an earlier Java web technology known as *applets*.

Directed Study

You should start this mini-project by reading *all* of Chapter 23 in Horstmann's *Big Java* (4th edition).

Pay special attention to Section 23.3 which introduces the concept of *managed beans*. Study the **`TimeZoneBean`** example and see if you can get this program up and running inside NetBeans.

You will also find it useful to work your way through the following tutorial:

http://NetBeans.org/kb/docs/web/jsf20-intro.html#requiredSoftware

All the exercises involve the use of the XHTML standard. If you are unfamiliar with this standard it is suggested that you look at one of the tutorials below

http://www.w3schools.com/html/html_xhtml.asp

http://www.freewebmasterhelp.com/tutorials/xhtml/

http://www.webheadstart.org/

Finally check out the **Useful Links** section at the end of this document.

Exercises

Exercise 1: Hello Illustrated World

In this exercise we use NetBeans (or some other IDE of your choice) to create a web application which utilises the Java Server Faces (JSF) framework. The application should display a graphical image containing a "Hello World!" message. When the browser first accesses the application the user will be presented with a button labelled **Get Message**. An easy way to do this is to modify the **index.xhtml** file which is automatically generated by NetBeans whenever a new Web Application project is created. The modification involves placing two JSF components in the body section of the file **index.xhtml**. The components required are:

- **h:form** : inserts an XHTML form element into a page
- **h:commandButton** : displays a button that triggers an event when clicked

Notice also that the title of the page has been set to **Hello World Test**

Figure 1 Viewing index.xhtml

When the button is clicked the following page is defined by a file called **helloWorld.xhtml** and sent back to the browser:

Figure 2 Displaying a Graphical Image

To build this simple web app use the JSF XHTML tag **h:graphicImage** as illustrated below:

```
<h:graphicImage library="images" name="earth.jpeg">
```

For this to work you need to set up a **resources** folder *within* your project's **Web Pages** node and then place an **images** folder inside resources. Finally store your graphics file inside **images**. Whilst you are doing this it would also be a good idea to create a folder called **css** to hold any CSS files you may wish to access at a later point in your laboratory work. Place this folder in the resources folder. See Appendix 1 for further information.

In NetBeans the application can be launched by right-clicking on the file **index.xhtml** and selecting the **Run File** menu item.

Notice that this file is stored under the **Web Pages** node in the project tab and you should do the same when creating XHTML files for use with your Web application.

To create a new XHTML file use the facilities provided by NetBeans.

- Right click on the **Web Pages** node
- Select **New --> XHTML**

This is illustrated below:

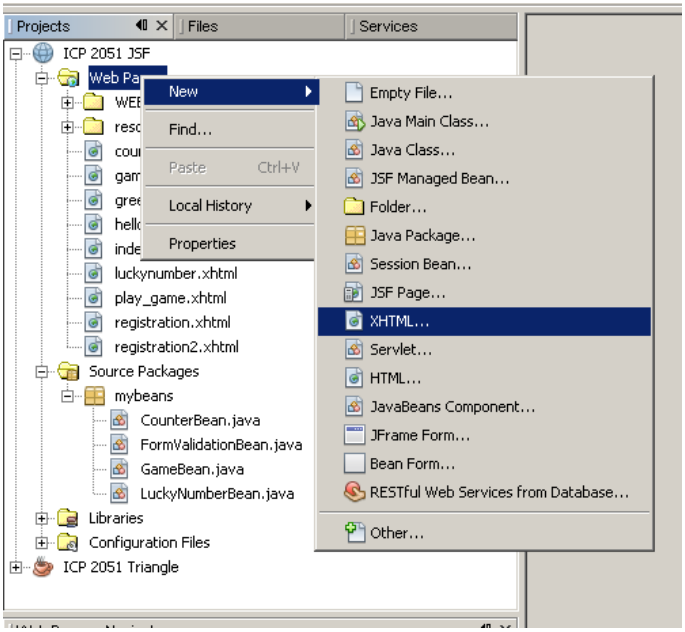

Figure 3 Using the Categories Menu

Notice that there are some other useful file types listed on this menu and you should make use of them where appropriate in the later exercises.

Exercise 2: Building a Lucky Number Bean

Create a JavaBean class called **`LuckyNumberBean`** to support a web application which generates so-called "lucky" numbers. These numbers are in fact random numbers (or more precisely pseudo-random numbers) between one and one million and as we shall see later can be used to build a simple online lottery. The front end of the application begins with an invitation to the user to play the game by generating an initial number.

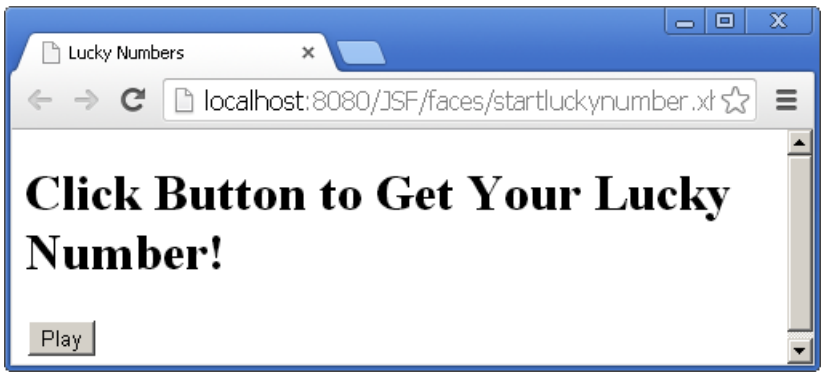

Figure 4 Display an Invitation to Play

The initial invitation is generated by a file called **startluckynumbers.xhtml**. When the button marked **Play** is clicked the screen below is generated.

Figure 5 Sample Output for the LuckyNumbersWeb Application

Clicking on the **Play** button will generate a new lucky number.

This exercise will involve getting to grips with JavaBean Components. You are strongly recommended to read Chapter 23 of *Big Java*. Pay particular attention to the definition of the **`TimeZoneBean`** class.

To assist development of JavaBean classes the NetBeans IDE provides a category **JSF Managed Bean** which can be obtained by selecting the package node in **Source Packages** and then:

New → Other → Java Server Faces → JSF ManagedBean

The file generated will contain the appropriate **`import`** statements and annotations.

Building this web application will involve creating three files:

- **startluckynumber.xhtml** – described above
- **luckynumber.xhtml** – this places the command button marked **Play** on the web page and displays the first "lucky" number generated by the user;
- **LuckyNumberBean.java** – this is a java program with a single method called **`getLuckyNumber()`** which returns an integer in the range [1, 1,000,000].

The key to this exercise is to write an appropriate JSF Expression Language expression (EL) which can interact with the JavaBean and thus obtain data from the **`LuckyNumberBean`** class.

Make sure that you know how to access the bean both from within NetBeans *and* directly using the address bar in the browser and entering a URL similar to the one below:

`http://localhost:8080/ICP_2151_JSF/faces/luckynumber.xhtml`

Having written this bean class you are now in a position to use the bean to construct a simple game of chance. This is the purpose of the next exercise.

Exercise 3: Session Tracking

Develop a web application which allows a user to play a simple game involving random numbers. The game involves the user guessing the value of a random number *before* it is generated. The application should

- Keep track of how many guesses have been entered;
- Display a message informing the player if they have won or lost.

When the application first starts the game counter for the session should be set to zero as shown in the screen shot below. The player then makes a guess and enters this value into the text box before clicking on **Play**.

Figure 6 The Initial Screen for Game Session

After the guess has been entered the application generates a lucky number and compares it with the number entered by the user. Win or lose a results page below should be generated by clicking **Play**.

Figure 7 Result and Game History Displayed

Note that the games counter is incremented every time the game is played during a particular session. Of course, if the player closes the connection with the server (i.e. by closing the browser rather than simply visiting another site) the counter will revert to zero when the game is next started.

This behaviour is achieved by using a **@SessionScoped** bean to maintain game data throughout the user session. The default scope of a JavaBean is request scope and thus to obtain behaviour illustrated in these screen shots you need to insert the **@SessionScoped** annotation into your bean class. I suggest that the name of the session bean should be **`GameBean`** and this Bean component will contain all the methods necessary to play this simple game.

Notice also that the pages generated by this Web application rely upon the use of JSF tags to produce text boxes, buttons and links. To obtain information about these tags and how to use them see Horstmann. Also check out the site below:

http://www.javabeat.net/2007/06/introduction-to-jsf-core-tags-library/

One final point: do not forget to test the methods in the bean class, in particular, find some way of checking that the method to determine whether a player has *won* or *lost* actually works. You may like to consider reducing the range (e.g. the lucky number is selected from only ten numbers) or alternatively you might find some way of manipulating the form parameters to ensure that the correct number is chosen.

Suggested file names for this web application:

- **GameBean.java**
- **play_game.xhtml** (Fig 6)
- **game_result.xhtml** (Fig 7)

One final point to check is that if another player accesses the application their games counter starts at zero. An easy way to do this is just to start up another browser – I used Firefox instead of Chrome.

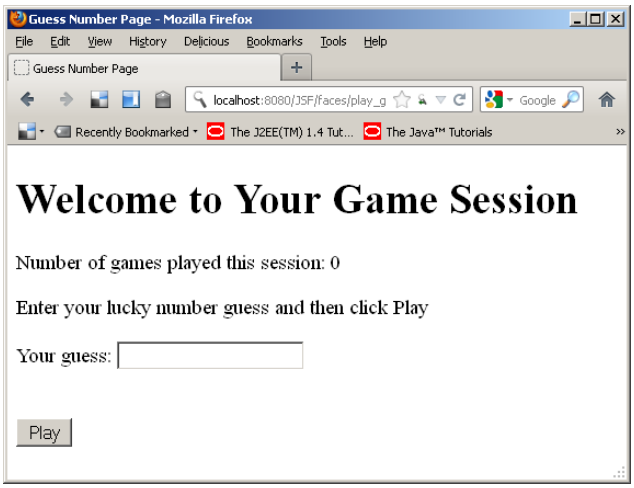

Figure 8 Starting a New Session from Firefox

Once you have got all this working congratulate yourself! You have mastered the basics of JSF.

Exercise 4: Creating Forms with JSF UI Components

Use an appropriate selection of JSF HTML tag library elements to produce a registration form for the Lucky Numbers game. Notice that I have placed a relevant graphical image on the page and you should do something similar.

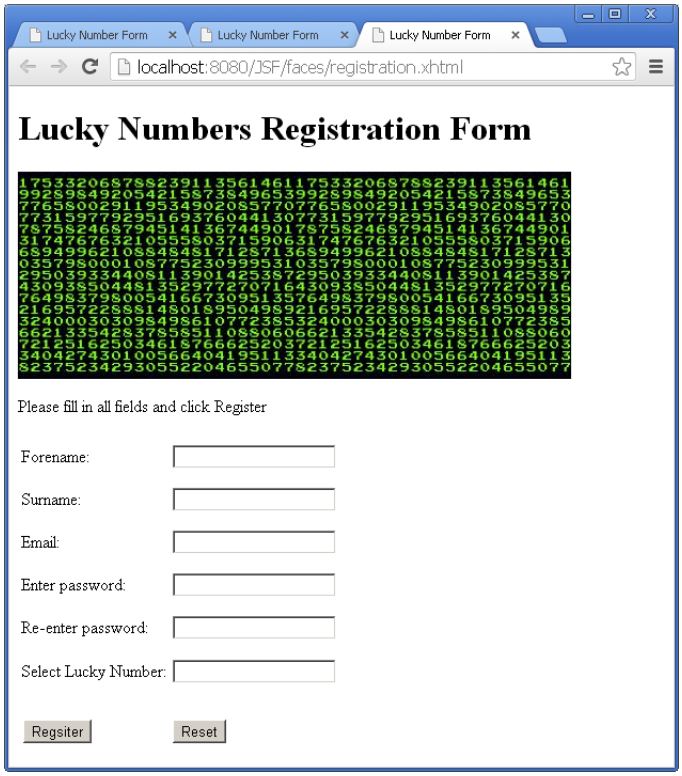

Figure 9 Form Created by registration.xhtml

Most of the tag library elements required for this exercise can be found in Section 23.5 of Horstmann's *Big Java.*

To help manage form layout try using the **<h:panelGrid>** tag which allows the web page designer to place JSF components in a grid structure.

An example of this tag is provided below:

```
<h:panelGrid columns="3" style="height: 96px; width:700px;">
```

The attributes in the above tag refer to the number of columns in the grid and the dimensions of the grid in pixels.

NB. This exercise is only concerned with form layout using JSF tag elements: there is no requirement to associate the **Register** button with a specific action but the **Reset** button should clear the text in all of the boxes. In addition, the validation of the form data is not a part of this exercise but is covered in Exercise 5.

Exercise 5: Validating Form Data

Part of the art of form design is to provide a mechanism for validating data elements as they are supplied by the user. To help the programmer with this task the JSF framework provides a rich set of validator elements including:

- **`f:validateLength`** – checks if field contains permitted number of characters
- **`f:validateDoubleRange`**– checks if the number is in range
- **`f:validateRequired`**– checks whether field contains value
- **`f:validateRegex`**–matches input against a regular expression
- **`f:validateBean`**–invoke a bean method to perform custom validation

Notice that these tags begin with **`f:`** so you will need to amend the html element description to include the attribute:

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

In this exercise you are asked to create a form which uses JSF standard validators to check the registration input for a user who wishes to register as a Lucky Numbers game player. The form is similar to the one produced in the previous exercise but in addition will display error messages and also store validated data in a bean class called **`FormValidationBean`**.

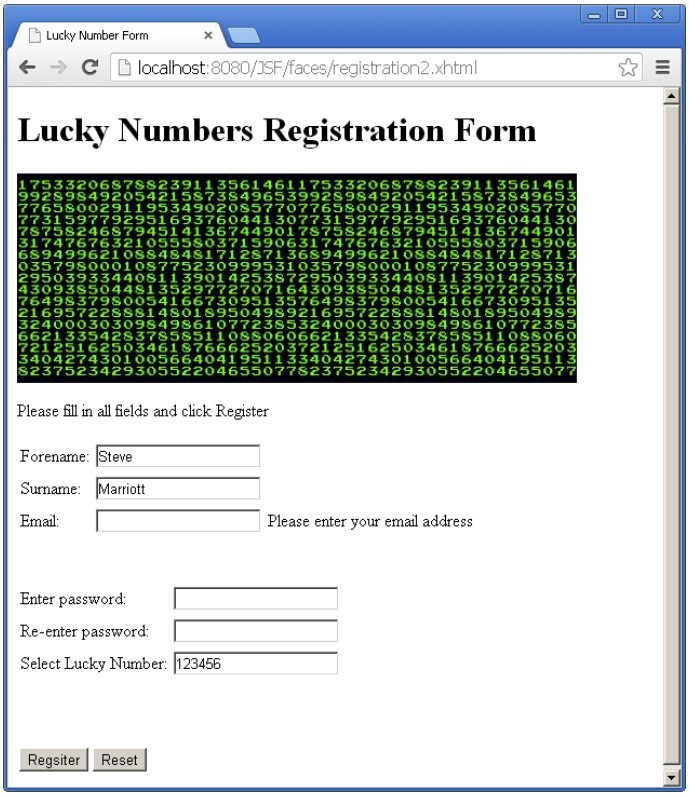

Figure 10 Validating User Input – register2.xhtml

The red text of the error message was obtained using a CSS style sheet. As usual the **Reset** button clears all text boxes.

Data validation will involve testing for the following:

- All text boxes are supplied with input
- Forenames and surnames contain no more than 30 characters
- The email supplied is a valid email address (i.e. as defined by a regular expression)
- The two password strings are identical
- The password entered is between 8 and 16 characters
- The lucky number is in the range [1, 1000000]

Validating the length of input strings (e.g. validating forenames and surnames) is best managed by using **`f:validateLength`** tags.

To validate the email address use a regular expression together with the tag **`f:validateRegex`**. You can, if you wish, devise your own regular expression for this task but why reinvent the wheel? A few minutes Internet research should yield a reliable regular expression to perform this validation.

Exercise 6: Building the Complete Game

Using JSF and the techniques illustrated in the previous exercises build a web application which allows users to

- Register a password and lucky number
- Log on to the site and be greeted with a personalised welcome message
- Play the game and get a result each time
- Obtain a play history recording the date and time of each play and the game result
- Log off

Note that the registered lucky number is the number used whenever the user logs on to the site and decides to play. In consequence the number chosen by the user must be stored in a database along with the user's password.

To do all of this properly would take us well beyond the scope of this course. For example, a genuine commercial system would not transmit unencrypted passwords However, with a little bit of thought and some research you ought to be able to get a "rough and ready" prototype up and working. Any reasonable effort will get a pretty good mark.

Remember to use **`SimpleDataSource`** to obtain a connection to the database. Also the database should be created independently of the web app by using SQL scripts through Workbench.

Useful Links

There is an excellent introduction to Java Server Faces in the textbook below:

Deitel H. and Deitel P., *Java™: How to Program* (9th edition), Pearson, 2012.

A good comprehensive introductory tutorial on JSF can be found at the site below

http://docs.oracle.com/javaee/1.4/tutorial/doc/JSFIntro.html

A NetBeans JSF tutorial can be found at

http://NetBeans.org/kb/docs/web/jsf20-intro.html

An introductory article on tags can be found at

http://www.javabeat.net/2007/06/introduction-to-jsf-core-tags-library/

A list of JSF tags can be found at

http://horstmann.com/corejsf/jsf-tags.html

Official reference documentation for JSF technology can be located at

http://java.sun.com/javaee/javaserverfaces/reference/api/index.html

Appendix 1: Project Folder Structure

After creating the resources folder and placing folders inside it my NetBeans project looked like the structure below. You should have something similar.

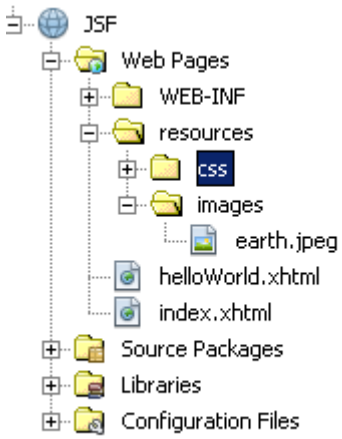

Figure 11 Example Folder Structure