



Managed Beans 2 – More Features

JSF 2.2 Version

Originals of slides and source code for examples: <http://www.coreservlets.com/JSF-Tutorial/jsf2/>

Also see the PrimeFaces tutorial – <http://www.coreservlets.com/JSF-Tutorial/primefaces/>
and customized JSF2 and PrimeFaces training courses – <http://courses.coreservlets.com/jsf-training.html>

Customized Java EE Training: <http://courses.coreservlets.com/>

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



**For live training on JSF 2, PrimeFaces, or other
Java EE topics, email hall@coreservlets.com**
Marty is also available for consulting and development support

Taught by the author of *Core Servlets and JSP*, this tutorial,
and JSF 2.2 version of *Core JSF*. Available at public venues, or
customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
 - JSF 2, PrimeFaces, Ajax, jQuery, Spring MVC, JSP, Android, general Java, Java 8 lambdas/streams, GWT, custom topic mix
 - Courses available in any location worldwide. Maryland/DC area companies can also choose afternoon/evening courses.
- Courses developed and taught by coreservlets.com experts (edited by Marty)
 - Hadoop, Spring, Hibernate/JPA, RESTful Web Services

Contact hall@coreservlets.com for details



Topics in This Section

- **Selecting values from choices: menus, list boxes, radio buttons**
- **Application scope**
- **Custom bean names**
- **Preselecting values**
- **Prepopulating input fields in general**

4

© 2015 Marty Hall



Selecting from Choices: Menus, Lists, and Radio Buttons



Customized Java EE Training: <http://courses.coreservlets.com/>

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Main Point

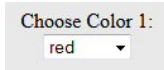
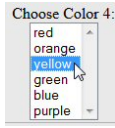
- **Textfields use single bean property**
 - Both the initial value (if non-empty) and the place where the value is sent upon submission
- **Menus and similar use two bean properties**
 - One provides the list of choices
 - The other is where the value is sent upon submission
 - Property can also affect initial selection, as shown later
- **Quick example**

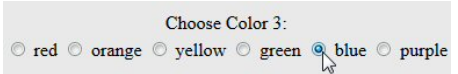
```
<h:selectOneMenu value="#{bean1.propertyForSelection}">
  <f:selectItems value="#{bean2.propertyForOptions}"/>
</h:selectOneMenu>
```

 - Similar for h:selectOneListbox and h:selectOneRadio
 - Can also use f:selectItem (singular) to provide single value. Will do this with Ajax, where f:selectItems provides real data, and f:selectItem provides placeholder of "Choose Value..."

6

Selection Input Types

- **h:selectOneMenu**
- **h:selectOneListbox**

Notice that it is h:selectOneListbox, not h:selectOneListBox
- **h:selectOneRadio**

If you use <h:selectOneRadio ... layout="pageDirection">, the radio buttons are arranged vertically. If you use layout="lineDirection" or (more commonly) omit the layout attribute, the radio buttons are arranged horizontally, as in the image.
- **Multi-selections**
 - Similar versions that allow multiple selections are h:selectManyMenu, h:selectManyListbox, and h:selectManyCheckbox.
 - For the selections, use array or List instead of single value
 - h:selectManyMenu is rendered very differently across browsers, and is hard to use in Firefox. Most people just forget h:selectManyMenu and use one of the other two.

7

Menus: More Details

```
<h:selectOneMenu value="#{bean1.selection}">  
  <f:selectItems value="#{bean2.options}" />  
</h:selectOneMenu>
```

submit to

list options

- **Beans**
 - Options and selection not necessarily come from same bean
- **Options**
 - Can be an array of values, a List of values, a Map (keys are display values, values are return values), or a group of beans (use itemLabel and itemVar)
- **The f: namespace**
 - Must be declared at the top of the page with
<html ... xmlns:f="http://xmlns.jcp.org/jsf/core" ...>

8

Options: More Details

- **Array of Strings (or other simple type)**
 - Values displayed and values returned are same
- **List of Strings**
 - Values displayed and values returned are same
- **Map<String,String>**
 - Map keys are the values displayed. One selection, the corresponding Map value is returned. You usually use LinkedHashMap to preserve ordering.
 - You can also have the Map values be complex objects, but then you need a converter.
- **Array or List of Objects**
 - Use itemLabel to choose part of Object to be displayed
 - Use itemValue to choose part of Object to be returned

9

More on Using Objects for Options

```
<h:selectOneListbox value="#{bean1.selection}">
  <f:selectItems value="#{bean2.options}"
    var="bn"
    itemLabel="#{bn.property1}"
    itemValue="#{bn.property2}"/>
</h:selectOneListbox>
```

- **var**
 - Made-up name. Each bean in the option will be bound to this name in turn.
- **itemLabel**
 - Part of bean to be displayed in the list of options
- **itemValue**
 - Part of bean to be returned upon user selection

10

Example: Input Page (Part 1)

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:f="http://xmlns.jcp.org/jsf/core"
  xmlns:h="http://xmlns.jcp.org/jsf/html">
```

Declare f: namespace so that f:selectItems can be used later in page

11

Example: Input Page (Part 2)

```
...
Choose Color 1:<br/>
<h:selectOneMenu value="#{colors1.color1}">
  <f:selectItems value="#{colorOptions.colorNames}"/>
</h:selectOneMenu>
...
Choose Color 2:<br/>
<h:selectOneListbox value="#{colors1.color2}">
  <f:selectItems value="#{colorOptions.colorValues}"/>
</h:selectOneListbox>
...
Choose Color 3:<br/>
<h:selectOneRadio value="#{colors1.color3}">
  <f:selectItems value="#{colorOptions.colorMap}"/>
</h:selectOneRadio>
...
```

String[]

List<String>

Map<String,String>

12

Example: Input Page (Part 3)

```
...
<h:selectOneListbox value="#{colors1.color4}">
  <f:selectItems value="#{colorOptions.colorList}"
    var="color"
    itemLabel="#{color.colorName}"
    itemValue="#{color.colorValue}"/>
</h:selectOneListbox>
...
<h:commandButton action="#{colors1.showColors}"
  value="Show Selected Colors"/>
...
```

List<Color>. Color has
getColorName and getColorValue.

13

Example: Java for Options

```
@ManagedBean // Also application scoped, discussed later
public class ColorOptions {
    private String[] colorNames =
        { "red", "orange", "yellow", "green", "blue", "purple" };
    private List<String> colorValues =
        Arrays.asList("#ff0000", "#ffa500", "#ffff00",
            "#008000", "#0000ff", "#800080");
    private Map<String,String> colorMap = new LinkedHashMap<>();
    private List<Color> colorList = new ArrayList<>();

    public ColorOptions() {
        for(int i=0; i<colorNames.length; i++) {
            String colorName = colorNames[i];
            String colorValue = colorValues.get(i);
            colorMap.put(colorName, colorValue);
            colorList.add(new Color(colorName, colorValue));
        }
    }
    // getColorNames, getColorValues, getColorMap, getColorList
}
```

14

Example: Color Class

```
public class Color {
    private final String colorName, colorValue;

    public Color(String colorName, String colorValue) {
        this.colorName = colorName;
        this.colorValue = colorValue;
    }

    public String getColorName() {
        return(colorName);
    }

    public String getColorValue() {
        return(colorValue);
    }
}
```

15

Example: Java for Selection

```
@ManagedBean
public class Colors1 {
    protected String color1, color2, color3, color4;

    // Getters and setters for the four properties
    // getColor1, setColor1, getColor2, setColor2,
    // getColor3, setColor3, getColor4, setColor4

    public String showColors() {
        return("show-colors-1");
    }
}
```

16

Example: Results Page

```
...
<h2>Color 1:
<span style="color: #{colors1.color1}">#{colors1.color1}
</span></h2>

<h2>Color 2:
<span style="color: #{colors1.color2}">#{colors1.color2}
</span></h2>

<h2>Color 3:
<span style="color: #{colors1.color3}">#{colors1.color3}
</span></h2>

<h2>Color 4:
<span style="color: #{colors1.color4}">#{colors1.color4}
</span></h2>
...
```

17

Example: Results

The screenshot shows a web browser window with the URL `localhost/managed-beans-2/selections-1.jsf`. The page title is "Selecting Elements from Choices". Below the title, it says "Examples of menus, list boxes, and radio buttons using arrays, lists, maps, and collections of Java objects."

There are four selection methods shown:

- h:selectOneMenu (Uses Array of Strings):** A dropdown menu labeled "Choose Color 1:" with "red" selected.
- h:selectOneListbox (Uses List of Strings):** A list box labeled "Choose Color 2:" with "#ffa500" selected.
- h:selectOneRadio (Uses Map):** Radio buttons labeled "Choose Color 3:" with "blue" selected.
- h:selectOneListbox (Uses List of POJOs):** A list box labeled "Choose Color 4:" with "purple" selected.

A "Show Selected Colors" button is located below the selection methods. To the right, a panel titled "Selected Colors" displays the results:

- Color 1: red
- Color 2: #ffa500
- Color 3: #0000ff
- Color 4: #800080

18

© 2015 Marty Hall



Application Scope



Customized Java EE Training: <http://courses.coreservlets.com/>

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Main Point

- **Bean scopes**
 - Control how long managed beans will stay “alive”, and which users and requests can access previous bean instances.
 - Specified with annotations or in faces-config.xml
- **Covered in this section**
 - Request scope, application scope
 - Plus brief summary of session scope
- **Covered in next section**
 - Session scope
 - Plus brief summary of all scopes
- **Covered in later sections**
 - Flow scope (JSF 2.2 only)
 - Declaring scopes in faces-config instead of with annotations

20

Request Scope: Interpretation

- **Meaning**
 - A new instance of the bean is created for every HTTP request, regardless of whether it is the same user or the same page. This is the most commonly used scope in all of JSF (session scope is second-most common).
- **Annotation**
 - @RequestScoped
 - But, request scope is default, so most developers simply omit the annotation

21

Request Scope: Quick Example

- **Java**

```
@ManagedBean  
public class BankForm { ... }
```

- **Facelets**

```
<h:inputText value="#{bankForm.customerId}"/>
```

- **Behavior**

- Bean is instantiated twice for each submission: once when form is displayed (and `getCustomerId` is called) and again when form is submitted (and textfield value is passed to `setCustomerId`).
- If same bean name appears on a different page, different instances are used.

22

Application Scope: Interpretation

- **Meaning**

- The bean is instantiated the first time any page with that bean name is accessed. From then on, the same bean instance is used, even if it is different user or different page. However, different Web apps are independent.
 - Never use application scope for beans that contain user data! Testing on your local machine with a single user might show correct results, but with multiple simultaneous users, you have race conditions with one user's data overwriting another's.

- **Annotation**

- `@ApplicationScoped`
 - Optionally, use with `@ManagedBean(eager=true)`
 - This option causes object to be instantiated when app loaded

23

Application Scope: Quick Example

- **Java**

```
@ManagedBean  
@ApplicationScoped  
public class Messages { ... }
```

- **Facelets**

```
{messages.message1}
```

- **Behavior**

- The first time this page (or any page with that bean name) is accessed, Messages is instantiated. From then on, the same bean instance is used for all users and on all pages that use that bean name.

24

Session Scope: Interpretation

- **Meaning**

- The bean is instantiated the first time any page with that bean name is accessed by a particular user. From then on, the same bean instance is used if it is same bean name and same user, even if it is different page. However, different users get different instances. User determined by JSESSIONID cookie (usually) or jsessionid URL parameter (sometimes).
 - Second-most common scope, after request scope.
 - Often used for user preferences (fonts, colors, languages). Also used for accumulating data over time (shopping carts, questions on exams).
 - The bean should be Serializable so that session data can live across server restarts and so that on clustered server, session data can be shared between nodes.

- **Annotation**

- @SessionScoped

25

Session Scope: Quick Example

- **Java**

```
@ManagedBean
@SessionScoped
public class UserLocale implements Serializable { ... }
```

- **Facelets**

```
<f:view locale="#{userLocale.selectedLanguage}"/>
```

- f:view and locales covered in section on event handling

- **Behavior**

- Bean is instantiated first time a particular user accesses any page that refers to that bean name.
- Same instance is used for that user on all pages that use that bean name.

26

Using Application Scope for Menus, Listboxes, etc.

- **In many apps, the options do not change**

- That is, the values listed by f:selectItems are always the same, and only the actual selections are specific to the user

- **If options can change**

```
<h:selectOneMenu value="#{bean1.propertyForSelection}">
  <f:selectItems value="#{bean1.propertyForOptions}"/>
</h:selectOneMenu>
```

Same user bean both times (not application scoped)

- **If options are always the same**

```
<h:selectOneMenu value="#{bean1.propertyForSelection}">
  <f:selectItems value="#{bean2.propertyForOptions}"/>
</h:selectOneMenu>
```

User data bean (not application scoped)

Options bean (application scoped)

- Using static data is also reasonable option

27

Previous Example: Java for Options

```
@ManagedBean
@ApplicationScoped
public class ColorOptions {
    private String[] colorNames =
        { "red", "orange", "yellow", "green", "blue", "purple" };
    private List<String> colorValues =
        Arrays.asList("#ff0000", "#ffa500", "#ffff00",
            "#008000", "#0000ff", "#800080");
    private Map<String,String> colorMap = new LinkedHashMap<>();
    private List<Color> colorList = new ArrayList<>();

    ...

    // getColorNames, getColorValues, getColorMap, getColorList
}
```

28

© 2015 Marty Hall



Custom Bean Names



Customized Java EE Training: <http://courses.coreservlets.com/>

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Main Point

- **The name attribute of @ManagedBean**

`@ManagedBean(name="anyName")`

`public class BeanName { ... }`

- You refer to bean with `#{anyName.blah}`, where bean name is exact value of name attribute

- **Scopes**

- Still request scoped by default
- Can still use any of the scopes discussed here and in later sections

30

Example

- **Idea (same behavior as previous example)**

- Choose colors in initial page
- Results page shows the selected colors

- **What changes from previous example**

- Java code gives custom name

`@ManagedBean(name="myBean")`

`public class Colors2 ... { ... }`

- Facelets pages refer to that name

`<h:selectOneMenu value="#{myBean.color1}">`

`<f:selectItems value="#{colorOptions.colorNames}"/>`

`</h:selectOneMenu>`

- **Unchanged**

- Java for options, Color class, functionality of Colors2

31

Example: Input Page (Excerpt)

```
...
Choose Color 1:<br/>
<h:selectOneMenu value="#{myBean.color1}">
  <f:selectItems value="#{colorOptions.colorNames}"/>
</h:selectOneMenu>
...
Choose Color 2:<br/>
<h:selectOneListbox value="#{myBean.color2}">
  <f:selectItems value="#{colorOptions.colorValues}"/>
</h:selectOneListbox>
...
Choose Color 3:<br/>
<h:selectOneRadio value="#{myBean.color3}">
  <f:selectItems value="#{colorOptions.colorMap}"/>
</h:selectOneRadio>
...
```

32

Example: Java for Selection

```
@ManagedBean(name="myBean")
public class Colors2 extends Colors1 {
    @Override
    public String showColors() {
        return("show-colors-2");
    }
}
```

33

Example: Results Page

```
...
<h2>Color 1:
<span style="color: #{myBean.color1}">#{myBean.color1}
</span></h2>

<h2>Color 2:
<span style="color: #{myBean.color2}">#{myBean.color2}
</span></h2>

<h2>Color 3:
<span style="color: #{myBean.color3}">#{myBean.color3}
</span></h2>

<h2>Color 4:
<span style="color: #{myBean.color4}">#{myBean.color4}
</span></h2>
...
```

34

Example: Results

The screenshot shows a web browser window with the URL `localhost/managed-beans-2/selections-2.jsf`. The page title is "Selecting Elements from Choices" and the subtitle is "Version 2: Using Custom Bean Names".

The page contains four color selection methods:

- h:selectOneMenu (Uses Array of Strings):** "Choose Color 1:" with a dropdown menu showing "green" selected.
- h:selectOneListbox (Uses List of Strings):** "Choose Color 2:" with a listbox showing "#800080" selected.
- h:selectOneRadio (Uses Map):** "Choose Color 3:" with radio buttons for red, orange, yellow, green, blue, and purple. The "blue" radio button is selected.
- h:selectOneMenu (Uses List of POJOs):** "Choose Color 4:" with a dropdown menu showing "red" selected.

A "Show Selected Colors" button is located below the selection methods.

Below the selection methods, a section titled "Selected Colors" displays the results:

- Color 1: green
- Color 2: #800080
- Color 3: #0000ff
- Color 4: #ff0000

35



Preselecting Choices for Menus, Listboxes, and Radio Buttons



Customized Java EE Training: <http://courses.coreservlets.com/>

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

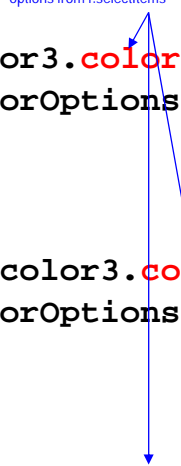
Main Idea

- **Getter method affects initial selection**
 - If getter method of user selection matches one of the choices, then that is the initially displayed value
 - If getter returns null or non-matching value
 - Menus: top value is shown
 - Others: no value is selected. Implication: for list boxes and radio buttons, always have getter match 1 of the choices
- **Maps and collections of objects**
 - Getter should match return value, not display value
- **Previous example lacked validation**
 - Form could have been submitted with no selection for the listboxes and the radio buttons
 - Validation covered in later lectures

Example: Input Page (Excerpt)

```
...
Choose Color 1:<br/>
<h:selectOneMenu value="#{color3.color1}">
  <f:selectItems value="#{colorOptions.colorNames}"/>
</h:selectOneMenu>
...
Choose Color 2:<br/>
<h:selectOneListbox value="#{color3.color2}">
  <f:selectItems value="#{colorOptions.colorValues}"/>
</h:selectOneListbox>
...
Choose Color 3:<br/>
<h:selectOneRadio value="#{color3.color3}">
  <f:selectItems value="#{colorOptions.colorMap}"/>
</h:selectOneRadio>
...
```

Corresponding getter returns value matching one of the options from f:selectItems



38

Example: Java for Selection

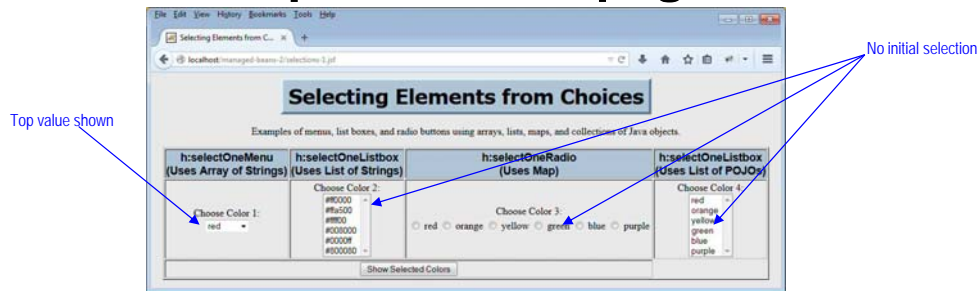
```
@ManagedBean
public class Colors3 extends Colors1 {
    public Colors3() {
        color1="orange";
        color2="#ff0000";
        color3="#008000";
        color4="#0000ff";
    }

    @Override
    public String showColors() {
        return("show-colors-3");
    }
}
```

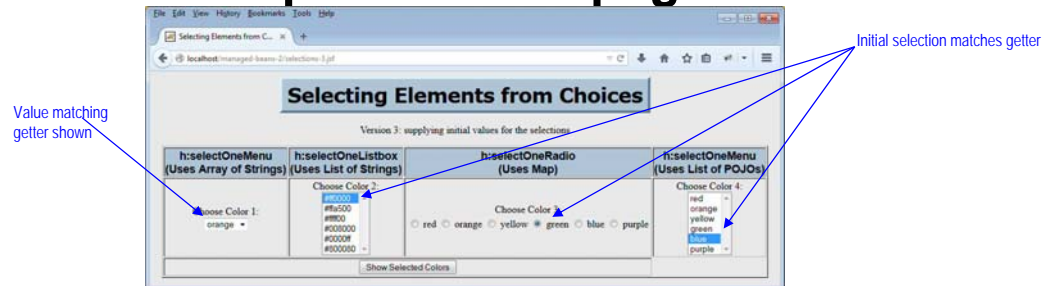
39

Example: Results

- First example on initial page load



- This example on initial page load



40

© 2015 Marty Hall



Prepopulating Other Fields



Customized Java EE Training: <http://courses.coreservlets.com/>

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Main Idea

- **Bean property refers to *both* getter & setter**
 - The getter method is called when form is displayed, and affects what is initially displayed to user
 - The value from the input element is passed to the setter method when the form is submitted
- **Examples**
 - `<h:inputText.../>` (textfield)
 - If getter returns non-empty (neither null nor ""), this is initial value inside textfield. Not used for password fields.
 - `<h:selectBooleanCheckbox.../>` (checkbox)
 - Getter returns true: initially checked. Otherwise unchecked
 - `<h:selectOneMenu.../>` (similar for listbox and radio btn)
 - If return value of getter matches an entry in menu, that is initial selection. Otherwise top entry is initially selected.

42

Textfields

- **Example 1 (Simple property)**
 - `<h:inputText value="#{someBean.someProperty}"/>`
 - When form initially displayed, call getSomeProperty. If value is something other than null or empty String, use it as initial value of textfield.
 - When form submitted, take the value in the textfield and pass it to setSomeProperty.
- **Example 2 (Chained properties)**
 - `<h:inputText value="#{someBean.a.b.c}"/>`
 - When form initially displayed, call getA, then getB on that result, then getC on that. If value of getC is non-empty, use it as initial value of textfield.
 - When form submitted, call getA, then getB on that result. Then take the value in the textfield and pass it to the setC method of that final result. **Only last one becomes setter.**

43

Checkboxes

- **Example**

- `<h:selectBooleanCheckbox value="#{someBean.someProperty}"/>`
 - When form initially displayed, call the accessor method, which must be of type boolean or Boolean. The name could be either isSomeProperty (most common) or getSomeProperty. If value is true, make checkbox initially checked. If value is false, make checkbox initially unchecked.
 - When form submitted, pass either true or false to setSomeProperty (which expects boolean or Boolean)
 - Note that JSF does not require anything (such as Struts 1.x does) to clear out old values of session-scoped checkboxes or radio buttons.

44

Drop Down Menus (And list boxes and radio buttons)

- **Example**

- `<h:selectOneMenu value="#{bean1.someProperty}">`
 `<f:selectItems value="#{bean2.choices}"/>`
 `</h:selectOneMenu>`
 - When form initially displayed, call `getChoices` (which should return List, array, or Map). This gives the entries in the drop down menu. Then call `getSomeProperty`. If result matches any of the entries, make that initially shown item. Otherwise use top item.
 - For menus, showing top entry is fine. But, with listboxes and radio buttons, if getter does not match a choice, nothing is selected when form comes up, and form could be submitted with no selection.
 - When form submitted, pass the current selection to `setSomeProperty`.

45

Example

- **Idea**
 - Collect input about programming background and preferences. Give recommended computer study plan.
- **Input form**
 - Textfield for favorite language. Prepopulate based on most popular language in the application
 - Drop down menu for second-favorite language. Make choices be the languages for which app has study guides. Make initial choice the second most popular language in the application.
 - Two checkboxes. Initial selection based on logic in app
- **Results page**
 - List of recommended languages. Requires looping tag.
 - Shown briefly here but covered in detail in later section

46

Input Form – Top (study-plan-input.xhtml)

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
...
</h:head>
```

Since we use the f:selectItems tag for the drop down menu, we must declare the f: namespace.

47

Input Form – Bottom (study-plan-input.xhtml)

```
<h:body>
...
<h:form>
  Email address:
  <h:inputText value="#{trainingForm.emailAddress}"/><br/>
  Favorite language:
  <h:inputText value="#{trainingForm.favoriteLanguage}"/><br/>
  Second favorite language:
  <h:selectOneMenu value="#{trainingForm.secondFavoriteLanguage}">
    <f:selectItems value="#{trainingForm.availableLanguages}"/>
  </h:selectOneMenu><br/>
  Programmed for 5+ years?
  <h:selectBooleanCheckbox value="#{trainingForm.expert}"/><br/>
  Personally know Larry Page, Steve Ballmer, and James Gosling?
  <h:selectBooleanCheckbox value="#{trainingForm.liar}"/><br/>
  <h:commandButton value="Show Recommended Study Plan"
    action="#{trainingForm.showTrainingPlan}"/>
</h:form>
...
</h:body></html>
```

Since getFavoriteLanguage returns non-empty (Java) originally, this textfield has an initial value when form displayed to user.

Since getSecondFavoriteLanguage returns a value that matches one of the choices in the list of languages, that option (JavaScript) is initially displayed to the user, rather than the top item being initially displayed.

isExpert initially returns true, so the first checkbox is initially checked.
isLiar initially returns false, so the second checkbox is initially unchecked.

48

Managed Bean (Part 1 – Properties for Input Elements)

```
@ManagedBean
public class TrainingForm {
  private String emailAddress;
  private String favoriteLanguage =
    LanguageUtils.findMostPopularLanguage(0);
  private String secondFavoriteLanguage =
    LanguageUtils.findMostPopularLanguage(1);
  private boolean isExpert = true;
  private boolean isLiar = false;

  // Getters and setters for all. The getters for
  // the boolean properties start with is, not get

  public String[] getAvailableLanguages() {
    return(LanguageUtils.languageArray());
  } // Options for dropdown box. See later slide.
```

49

Managed Bean (Part 2 – Action Controller)

```
public String showTrainingPlan() {  
    int numLanguagesToStudy;  
    if (isExpert) {  
        numLanguagesToStudy = 4;  
    } else {  
        numLanguagesToStudy = 2;  
    }  
    if (isLiar) {  
        return("liar");  
    } else {  
        languagesToStudy =  
            LanguageUtils.randomLanguages(numLanguagesToStudy);  
        return("study-plan");  
    }  
}
```

This List is the placeholder that is filled in. Since it can be of varying length, the results page will need to use a loop.

50

Managed Bean (Part 3 – Placeholder for Results)

```
private List<String> languagesToStudy;  
  
public List<String> getLanguagesToStudy() {  
    return(languagesToStudy);  
}
```

51

Code for Options in Dropdown

- **Main managed bean**

```
public String[] getAvailableLanguages() {  
    return(LanguageUtils.languageArray());  
}
```

- **Helper class (LanguageUtils)**

```
public class LanguageUtils {  
    private static String[] languages =  
        { "Java", "JavaScript", "C#", "C++", "PHP", "Python",  
          "Perl", "Ruby", "Scala" };  
  
    ...  
  
    public static String[] languageArray() {  
        return(languages);  
    }  
  
    ...  
}
```

The basic idea of dropdown menus is that the value of `f:selectItems` points at an array, List, or Map. If array or List, values displayed are same as values returned on selection. If Map, keys are displayed to user but corresponding Map values are returned on selection. (Use `LinkedHashMap` to maintain order.) You can also use the legacy `SelectItem` class from JSF 1, but there is no reason to in JSF 2.

52

Main Results Page (study-plan.xhtml)

```
<!DOCTYPE ...>  
<html xmlns="http://www.w3.org/1999/xhtml"  
      xmlns:h="http://xmlns.jcp.org/jsf/html"  
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">  
  <h:head>...</h:head>  
  <h:body>  
  ...  
  <ul>  
    <ui:repeat var="language"  
              value="#{trainingForm.languagesToStudy}">  
      <li>#{language}</li>  
    </ui:repeat>  
  </ul>  
  ...  
</h:body></html>
```

There is a whole separate tutorial section on looping and handling variable-length data. But the basic idea is simple: almost identical to the JSTL `c:forEach` loop and very similar to the Java `for(Blah b: collectionOfBlahs)` loop.

53

“Error” Page (liar.xhtml)

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>...</h:head>
<h:body>
...
<h2>Yeah, right. And you know more JavaScript than
Brendan Eich or Doug Crockford, I suppose?</h2>
<p>Please <a href="study-plan-input.jsf">try again</a>,
but be honest this time.</p>
...
</h:body></html>
```

Insulting your users is not generally considered a recommended practice.

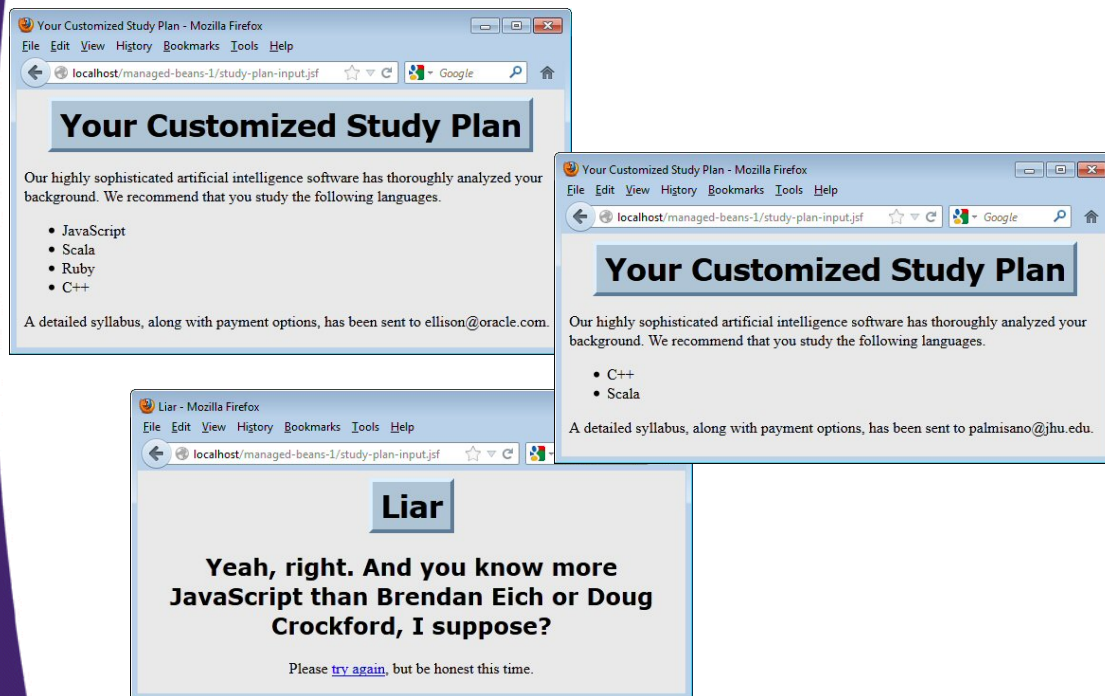
54

Results (Input Form)

The screenshot shows a Mozilla Firefox browser window with the title 'Create Customized Study Plan - Mozilla Firefox'. The address bar shows 'localhost/managed-beans-1/study-plan-input.jsf'. The page content includes a heading 'Create Customized Study Plan', a paragraph 'Our highly sophisticated artificial intelligence software will create a customized study plan for the programming languages you want to learn.', and a section titled 'Enter Background and Interests'. This section contains the following form elements: 'Email address:' with a text input field, 'Favorite language:' with a text input field containing 'Java', 'Second favorite language:' with a dropdown menu showing 'JavaScript', 'Programmed for 5+ years?' with a checked checkbox, 'Personally know Larry Page, Steve Ballmer, and James Gosling?' with an unchecked checkbox, and a 'Show Recommended Study Plan' button.

55

Results (Results Pages)



56

© 2015 [Marty Hall](#)



Wrap-Up



Customized Java EE Training: <http://courses.coreservlets.com/>

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Summary

- **Selecting values from choices**

```
<h:selectOneMenu value="#{bean1.propertyForSelection}">  
  <f:selectItems value="#{bean2.propertyForOptions}" />  
</h:selectOneMenu>
```

- **Application scope**

- The bean for the options can use @ApplicationScoped

- **Custom bean names**

- @ManagedBean(name="customName")

- **Preselecting values**

- If `getPropertyForSelection()` matches one of `getPropertyForOptions()`, that is initial selection

- **Prepopulating input fields in general**

- Getter method affects what is initially shown.

58

© 2015 Marty Hall



Questions?

More info:

<http://www.coreservlets.com/JSP-Tutorial/jsp2/> – JSP 2.2 tutorial

<http://www.coreservlets.com/JSP-Tutorial/primefaces/> – PrimeFaces tutorial

<http://courses.coreservlets.com/jsf-training.html> – Customized JSF and PrimeFaces training courses

<http://coreservlets.com/> – JSF 2, PrimeFaces, Java 7 or 8, Ajax, jQuery, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training



Customized Java EE Training: <http://courses.coreservlets.com/>

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.