

# JSF Programming Basics

- 1 Simplified flow of control
- 2 ManagedBeans
- 3 Action controller

# JSF Page Structure

```
1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5     xmlns:h="http://xmlns.jcp.org/jsf/html">
6   <h:head>
7     <title>Facelet Title</title>
8   </h:head>
9   <h:body>
10    <h:form>
11      <!-- code -->
12    </h:form>
13
14    <h:form>
15      <!-- code -->
16    </h:form>
17
18    <h:form>
19      <!-- code -->
20    </h:form>
21  </h:body>
22 </html>
```

# Managed Beans

1. They are usually POJOs (they implement no special interfaces, and most methods have no JSF-specific argument or return types).

2. They have *setters* and *getters* for properties

3. They have an *actionControllerMethod* that takes no arguments and returns a String (or, in general, an Object whose *toString()* is used). This is the method listed in the action of the `h:commandButton` in the input form.

4. You refer to the bean using `#{someBean.something}`

```
1 package edu.slcc.asdv.beans;
2
3 import javax.inject.Named;
4 import javax.enterprise.context.RequestScoped;
5
6 @Named(value = "someBean")
7 @RequestScoped
8 public class SomeBean
9 {
10     private String someProperty;
11
12     public SomeBean()
13     {
14     }
15     public String getSomeProperty()
16     {
17         return someProperty;
18     }
19
20     public void setSomeProperty(String someProperty)
21     {
22         this.someProperty = someProperty;
23     }
24     public String actionControllerMethod()
25     {
26         return "something";
27     }
28 }
```

# Practice 1

Click on button in initial page and get one of three results pages, chosen at random

## **Create a JSF starting page(index)**

```
<h:commandButton...action="#{navigator.choosePage}"/>
```

## **Create a Managed Bean**

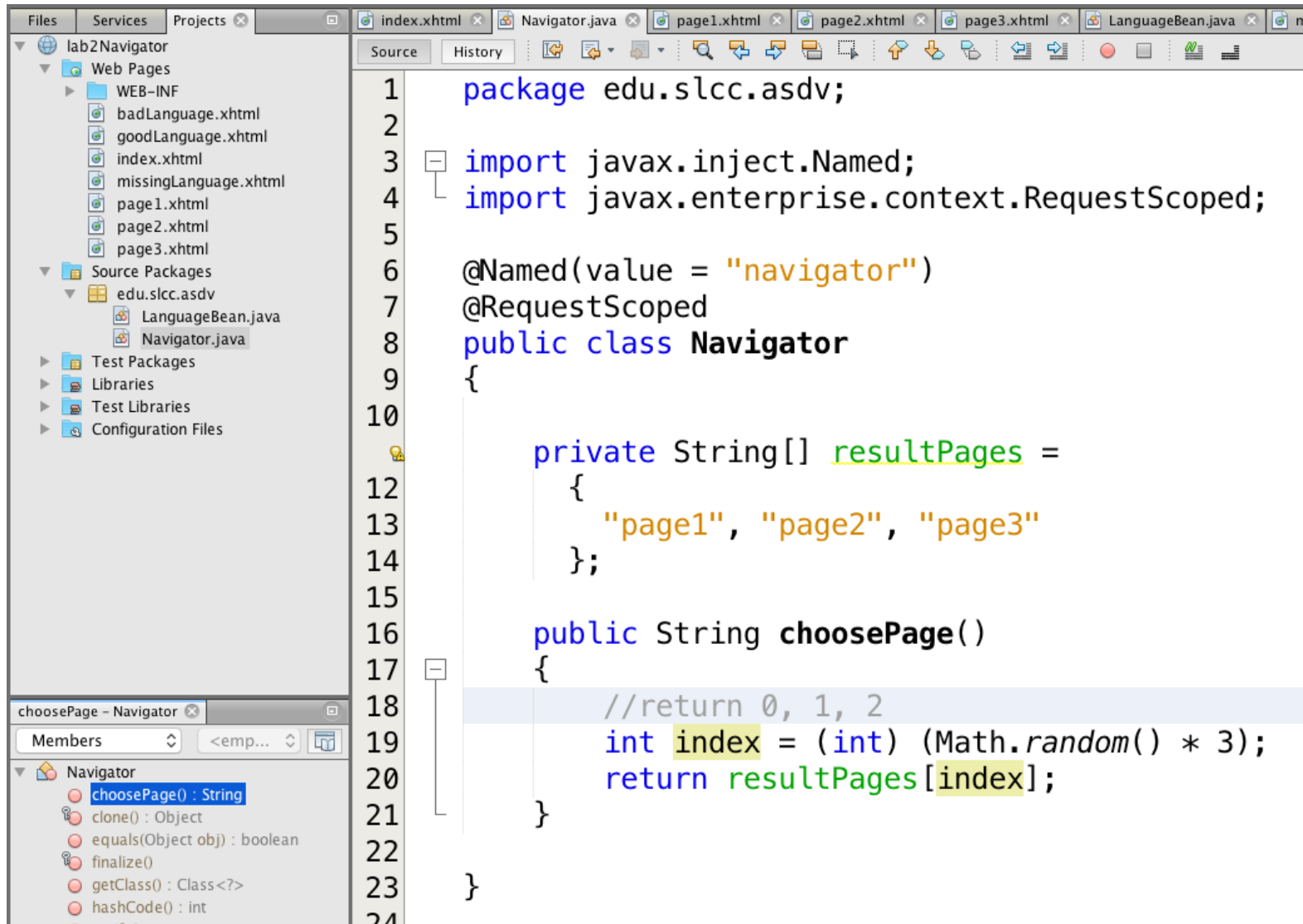
Action controller method returns 3 possible Strings

– "page1", "page2", or "page3"

## **Create 3 result pages**

page1.xhtml, page2.xhtml, and page3.xhtml

# Create the Managed Bean --Navigator



The screenshot shows an IDE window with the following code in `Navigator.java`:

```
1 package edu.slcc.asdv;
2
3 import javax.inject.Named;
4 import javax.enterprise.context.RequestScoped;
5
6 @Named(value = "navigator")
7 @RequestScoped
8 public class Navigator
9 {
10
11     private String[] resultPages =
12     {
13         "page1", "page2", "page3"
14     };
15
16     public String choosePage()
17     {
18         //return 0, 1, 2
19         int index = (int) (Math.random() * 3);
20         return resultPages[index];
21     }
22
23 }
24
```

The IDE interface also shows a project explorer on the left with the following structure:

- lab2Navigator
  - Web Pages
    - badLanguage.xhtml
    - goodLanguage.xhtml
    - index.xhtml
    - missingLanguage.xhtml
    - page1.xhtml
    - page2.xhtml
    - page3.xhtml
  - Source Packages
    - edu.slcc.asdv
      - LanguageBean.java
      - Navigator.java
  - Test Packages
  - Libraries
  - Test Libraries
  - Configuration Files

At the bottom, a 'Members' window for the `Navigator` class is visible, showing the `choosePage() : String` method.

# Implementation of JSF Page

```
1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5     xmlns:h="http://xmlns.jcp.org/jsf/html">
6   <h:head>
7     <title>Facelet Title</title>
8   </h:head>
9   <h:body>
10    <h:form>
11      Press button to get one of three possible results pages.
12      <br/>
13      <h:commandButton value="Go to Random Page"
14        action="#{navigator.choosePage}" />
15    </h:form>
16  </h:body>
17 </html>
```

# Output

South Louisiana Co... Canvas | South Loui... Gmail Ειδήσεις

Press button to get one of three possible results pages.

Go to Random Page

South Louisiana Co... Canvas | South Loui...

**One, Uno, Ένα**

South Louisiana Co... Canvas | South Loui...

**Three, Tres, Τρία**

South Louisiana Co... Canvas | South Loui...

**Two Duo Δύο**

# Practice 2

## Parameter Passing – h:InputText

- We will BIND **h:inputText** to a property of bean – setter getter
- The user will enter a language that CAN be used in JSF. Depending on what the user enter in input-text we will navigate him to the proper JSF page



# Create the LanguageBean

The screenshot shows an IDE window with the following components:

- Files Panel (Left):** Shows a project structure for 'lab2Navigator'. Under 'Web Pages', there are files 'index.xhtml', 'page1.xhtml', 'page2.xhtml', and 'page3.xhtml'. Under 'Source Packages', there is a package 'edu.slcc.asdv' containing 'LanguageBean.java' and 'Navigator.java'. There are also 'Test Packages', 'Libraries', 'Test Libraries', and 'Configuration Files'.
- Code Editor (Center):** Displays the source code for 'LanguageBean.java'. The code is as follows:

```
1 package edu.slcc.asdv;
2
3 import javax.inject.Named;
4 import javax.enterprise.context.RequestScoped;
5
6 @Named(value = "languageBean")
7 @RequestScoped
8 public class LanguageBean
9 {
10
11     private String language;
12
13     public String getLanguage()
14     {
15         return language;
16     }
17
18     public void setLanguage(String language)
19     {
20         this.language = language.trim();
21     }
22 }
23
```
- Members Panel (Bottom):** Shows the members of the 'LanguageBean' class, including 'clone() : Object', 'equals(Object obj) : boolean', 'finalize()', 'getClass() : Class<?>', and 'getLanguage() : String'.

# Add 2 more methods to LanguageBean

```
23     public String showChoice()  
24     {  
25  
26         if (language.equalsIgnoreCase("Java")  
27             || language.equalsIgnoreCase("Groovy"))  
28             {  
29                 return ("goodLanguage");  
30             }  
31         else if (isMissing(language))  
32             {  
33                 return ("missingLanguage");  
34             }  
35         else  
36             {  
37                 return ("badLanguage");  
38             }  
39     }  
40  
41     private boolean isMissing(String value)  
42     {  
43         return ((value == null) || (value.trim().isEmpty()));  
44     }  
45 }  
46
```

# Add form to index page lines 11 to 17

```
8      </h:head>
9      <h:body>
11     <h:form>
12         Enter a programming language and i will tell you if it
13         can be used to implement JSF managed beans:<br/>
14         <h:inputText value="#{languageBean.language}"/><br/>
15         <h:commandButton value="Check Language"
16             action="#{languageBean.showChoice}"/>
17     </h:form>
18
19     <br/>
20     <h:form>
21         Press button to get one of three possible results pages.
22         <br/>
23         <h:commandButton value="Go to Random Page"
24             action="#{navigator.choosePage}"/>
25     </h:form>
26 </h:body>
27 </html>
```

# Add JSF missingLanguage to Web Pages

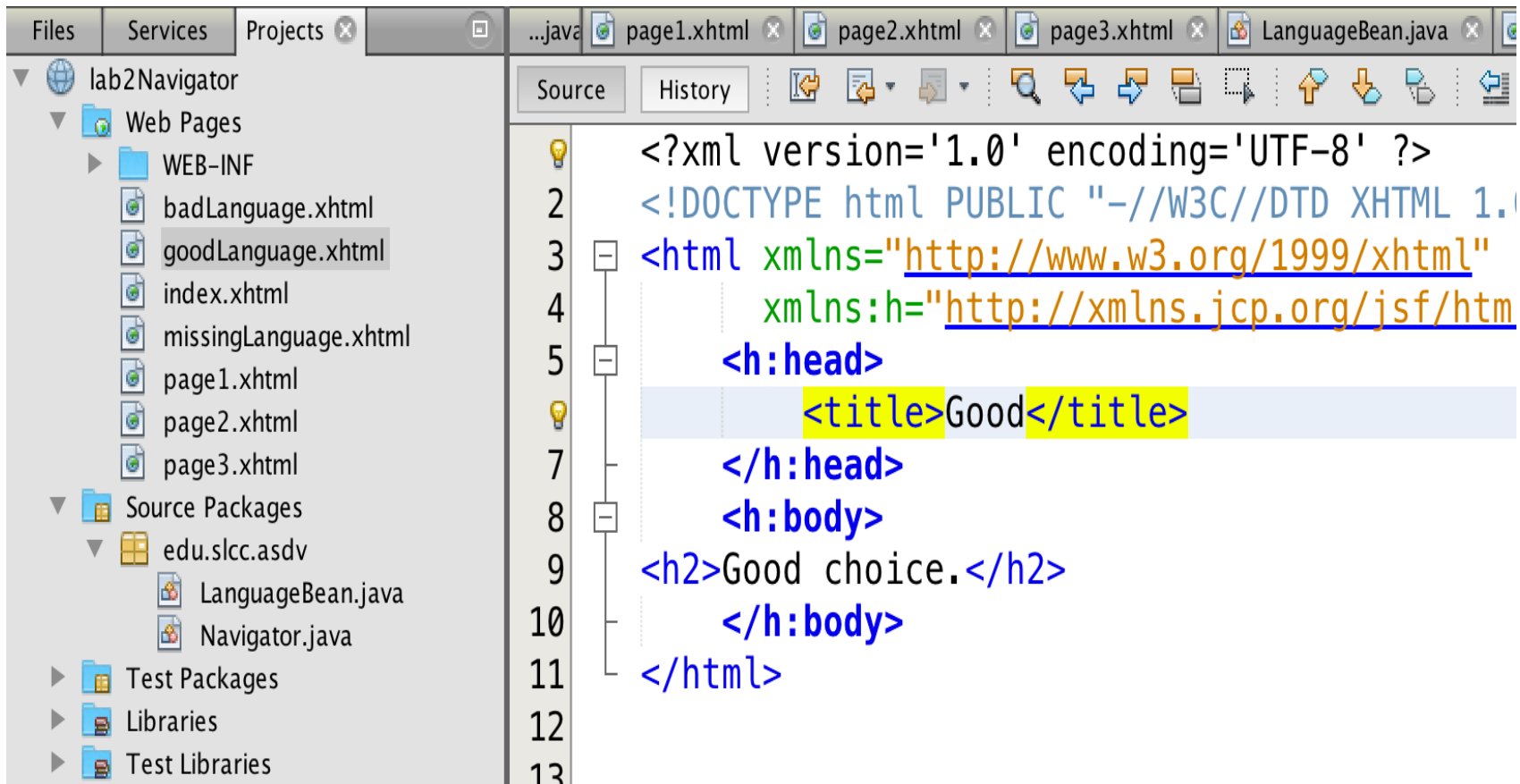
The screenshot shows an IDE window titled 'lab2Navigator'. The left sidebar (Navigator) displays a project structure:

- lab2Navigator
  - Web Pages
    - WEB-INF
    - badLanguage.xhtml
    - goodLanguage.xhtml
    - index.xhtml
    - missingLanguage.xhtml
    - page1.xhtml
    - page2.xhtml
    - page3.xhtml
  - Source Packages
    - edu.slcc.asdv
      - LanguageBean.java
      - Navigator.java
  - Test Packages
  - Libraries
  - Test Libraries
  - Configuration Files

The main editor shows the source code of 'missingLanguage.xhtml' with line numbers 1 through 17. The code is as follows:

```
1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Trans:
3 <html xmlns="http://www.w3.org/1999/xhtml"
4     xmlns:h="http://xmlns.jcp.org/jsf/html">
5   <h:head>
6     <title>Ugly</title>
7   </h:head>
8   <h:body>
9     <table class="title">
10      <tr><th>Missing Language</th></tr>
11    </table>
12    <h2>Duh! You didn't enter a language!
13      (<a href="index.xhtml">Try again</a>)
14    </h2>
15  </h:body>
16 </html>
```

# Add JSF goodLanguage to WebPages



The screenshot shows an IDE window with a project explorer on the left and a code editor on the right. The project explorer shows a project named 'lab2Navigator' with a 'Web Pages' folder containing several XHTML files, including 'goodLanguage.xhtml'. The code editor displays the content of 'goodLanguage.xhtml', which is an XHTML document. The document includes XML declarations, a DOCTYPE, and HTML elements for the head and body. The title 'Good' is highlighted in yellow, and the body contains an h2 element with the text 'Good choice.'

```
1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml"
4       xmlns:h="http://xmlns.jcp.org/jsf/html"
5       >
6     <h:head>
7       <title>Good</title>
8     </h:head>
9     <h:body>
10      <h2>Good choice.</h2>
11    </h:body>
12  </html>
```

# Add JSF badLanguage to Web Pages

```
1  <?xml version='1.0' encoding='UTF-8' ?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Tran
3  <html xmlns="http://www.w3.org/1999/xhtml"
4      xmlns:h="http://xmlns.jcp.org/jsf/html">
5      <h:head>
6          <title>Bad</title>
7      </h:head>
8      <h:body>
9          <h2>Use #{languageBean.language} in JSF?
10         Be serious!</h2>
11      </h:body>
12 </html>
```