CMP 436/774

Introduction to Java Server faces (JSFs) -Part 1

Fall 2012
Department of Mathematics
and Computer Science
Lehman College, CUNY

Java Server Faces (JSF) Introduction

References

- **Chapters 4-12 (of the reference R1: Java EE 6 Tutorial)**
- **Chapters 4, 5 (of the reference 5:** David R Heffelfinger , **Java EE 6 Development with NetBeans 7**)
- JSF 2.0 from reference 4 (from www.coreservlets.com)
- Other references
  - http://netbeans.org/kb/docs/web/jsf20-support.html?print=yes
  - And other URLs (updated)

3

Java Server Faces (1)

- **Most Java Web applications are developed using well known web application frameworks such as Apache Struts, Spring, and other MVC architectures.**
  - These frameworks are built based on top of the JSP/Servlet technologies.
  - Many web application frameworks are available (more than 30)
- **JSF can be considered as web application frameworks in Java EE specification.**
  - It's better considered as a **server-side component framework**.
  - JSF can be used to write applications that are not web based
    - But mainly used for web applications
- **JSF application consists of a series of XHTML pages containing custom JSF tags, one or more JSF managed beans, optional configuration file named faces-config.xml**

4

JSF (2)

JavaServer Faces technology consists of the following:

- An API for
  - representing components and managing their state
  - handling events, server-side validation, and data conversion
  - defining page navigation
  - supporting internationalization and accessibility
  - providing extensibility for all these features
- Tag libraries for adding components to web pages and for connecting components to server-side objects

5

JSF (3)

FacesServlet is a servlet that manages the request processing lifecycle for web applications that are utilizing JSFs to construct the user interface.

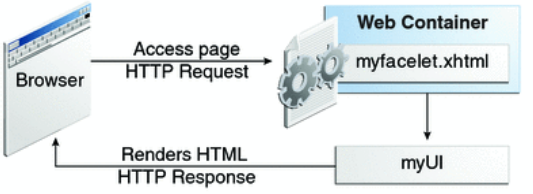
JSF Code can be reused and extended for components through the templating and composite component features.

- By JSF Annotations, you can automatically register the managed bean as a resource available for JSF applications.
- Implicit navigation rules allow developers to quickly configure page navigation (via faces-config.xml).
- JSF technology provides a rich architecture for managing component state, processing component data, validating user input, and handling events.

6

Client Server Interaction in JSF Application

- The web page, myfacelet.xhtml, is built using JavaServer Faces component tags. Component tags are used to add components to the view (represented by myUI in the diagram), which is the server-side representation of the page.
- In addition to components, the web page can also reference objects, such as the following:
  - Any event listeners, validators, and converters that are registered on the components
  - The JavaBeans components that capture the data and process the application-specific functionality of the components



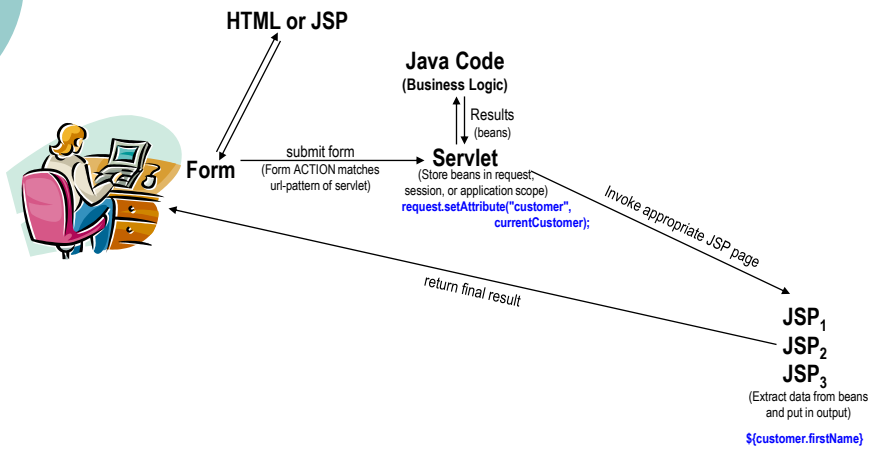
7

JSF vs JSP/Servlet

- **MVC**
  - A fair comparison is to look at the use of the JSF framework vs. the use of MVC with servlets and JSP

8

A Quick Review of Standard MVC (JSP/Servlet)



9

Advantages of JSF vs. Standard MVC (1)

- **Custom GUI controls**
  - JSF provides a set of APIs and associated custom tags to create HTML forms that have complex interfaces
    - There are many extra-rich third-party JSF libraries
- **Event handling**
  - JSF makes it easy to designate Java code that is invoked when forms are submitted. The code can respond to particular buttons, changes in particular values, certain user selections, and so on.
- **Managed beans**
  - In JSP, you can automatically populate a bean based on request parameters. JSF extends this capability and adds in several utilities, all of which serve to greatly simplify request param processing.
- **Integrated Ajax support**
  - You can use jQuery, Dojo, or Ext-JS with servlets and JSP. However, JSF lets you use Ajax without explicit JavaScript programming and with very simple tags.

10

Advantages of JSF vs. Standard MVC (2)

- **Form field conversion and validation**
  - JSF has builtin capabilities for checking that form values are in the required format and for converting from strings to various other data types. If values are missing or in an improper format, the form can be automatically redisplayed with error messages and with the previously entered values maintained.
- **Page templating**
  - Although JSP has jsp:include for reuse of content and jspf, JSF has a full-fledged page templating system that lets you build pages that share layout or content
- **Consistent approach**
  - JSF encourages consistent use of MVC throughout your application.

Disadvantages of JSF vs. Standard MVC (1)

- **Bigger learning curve**
  - To use MVC with the standard RequestDispatcher, you need to be comfortable with the standard JSP and servlet APIs. To use MVC with JSF, you have to be comfortable with the servlet API *and* a large and elaborate framework that is almost equal in size to the core system.
  - Similarly, if you have an existing app and want to add in some small amounts of Ajax functionality, it is moderately easy with jQuery (quite easy if you know JavaScript already). Switching your app to JSF 2.0 is a big investment.
- **Worse documentation**
  - Compared to the standard servlet and JSP APIs, JSF has fewer online resources, and many developers find the online JSF documentation confusing and often poorly organized.

Disadvantages of JSF vs. Standard MVC (2)

- **Less transparent**
  - With JSF applications, there is a lot more going on behind the scenes than with normal JSP/Servlet Web applications. As a result, JSF applications are:
    - Harder to understand
    - Harder to benchmark and optimize
- **Rigid approach**
  - The flip side of the benefit that JSF encourages a consistent approach to MVC is that JSF makes it difficult to use other approaches.

JSF example (hello.xhtml)

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

<h:head>
      <title>Hello World</title>
</h:head>
<h:body>
      <h3>Hello World Example - hello.xhtml</h3>
      <h:form>
            <h:inputText value="#{helloBean.name}"></h:inputText>
            <h:commandButton value="Welcome Me" action="welcome"></h:commandButton>
      </h:form>
</h:body>
</html>
```

you can put the page name directly in the button's "action" attribute. For simple navigation, it's more than enough, but, for complex navigation, you need to use the "navigation rule" in "faces-config.xml".

hello.xhtml – Renders a JSF text box and link it with the "helloBean" (JSF managed bean), "name" property, and also a button to display the "welcome.xhtml" page when it's clicked.

welcome.xhtml

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

   <h:head>
      <title>Hello World</title>
   </h:head>
   <h:body bgcolor="white">
      <h3>Hello World Example - welcome.xhtml</h3>
      <h4>Welcome #{helloBean.name}</h4>
   </h:body>
</html>
```

The #{…} indicate this is a JSF expression language (EL) #{helloBean.name}, when the page is submitted, JSF will find the "helloBean" and set the submitted textbox value via the setName() method. When welcome.xhtml page is display, JSF will find the same session "helloBean" again and display the name property value via the getName() method.

15

Managed HelloBean

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean
@SessionScoped
public class HelloBean {

        private String name;

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

}
```

JSF managed bean, with a name property to store user data.

name property can be accessed from a JSF page.

16

web.xml (1)

```
<!-- Change to "Production" when you are ready to deploy -->
        <context-param>
                <param-name>javax.faces.PROJECT_STAGE</param-name>
                <param-value>Development</param-value>
        </context-param>

        <!-- Welcome page -->
        <welcome-file-list>
                <welcome-file>faces/hello.xhtml</welcome-file>
        </welcome-file-list>

        <!-- JSF mapping -->
        <servlet>
                <servlet-name>Faces Servlet</servlet-name>
                <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
                <load-on-startup>1</load-on-startup>
        </servlet>
```


web.xml (2)

```
<!-- Map these files with JSF -->
        <servlet-mapping>
                <servlet-name>Faces Servlet</servlet-name>
                <url-pattern>/faces/*</url-pattern>
        </servlet-mapping>
        <servlet-mapping>
                <servlet-name>Faces Servlet</servlet-name>
                <url-pattern>*.jsf</url-pattern>
        </servlet-mapping>
        <servlet-mapping>
                <servlet-name>Faces Servlet</servlet-name>
                <url-pattern>*.faces</url-pattern>
        </servlet-mapping>
        <servlet-mapping>
                <servlet-name>Faces Servlet</servlet-name>
                <url-pattern>*.xhtml</url-pattern>
        </servlet-mapping>
</web-app>
```

helloAjax.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html">
  <h:body>
        <h3>Ajax Hello World Example</h3>
        <h:form>
                <h:inputText id="name" value="#{helloBean.name}"></h:inputText>
                <h:commandButton value="Welcome Me">
                        <f:ajax execute="name" render="output" />
                </h:commandButton>

                <h2><h:outputText id="output" value="#{helloBean.sayWelcome}" /></h2>
        </h:form>
  </h:body>
</html>
```

In this example, it make the button Ajaxable. When the button is clicked, it will make an Ajax request to the server instead of submitting the whole form.

In the <f:ajax> tag :
execute="name" – Indicate the form component with an Id of "name" will be sent to the server for processing. For multiple components, just split it with a space in between, e.g execute="name anotherId anotherxxId". In this case, it will submit the text box value.
render="output" – After the Ajax request, it will refresh the component with an id of "output". In this case, after the Ajax request is finished, it will refresh the <h:outputText> component.

19

Hellobean.java

```
@ManagedBean
@SessionScoped
public class HelloBean {
        private String name;

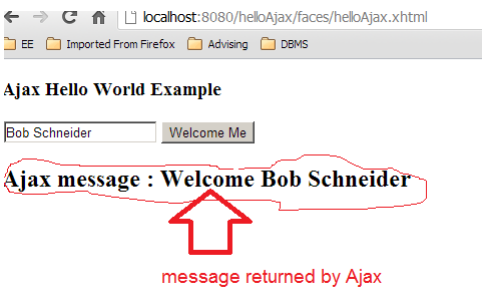
        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        public String getSayWelcome(){
                //check if null?
                if("".equals(name) || name ==null){
                        return "";
                }else{
                        return "Ajax message : Welcome " + name;
                }
        }
}
```

20

Run helloAjax



When the button is clicked, it makes an Ajax request and pass the text box value to the server for processing. After that, it refreshes the outputText component and displays the value via getSayWelcome() method, without any "page flipping effect".

21

Facelets

- **Facelets is a powerful but lightweight page declaration language that is used to build JSF views using HTML style templates and to build component trees.**
- **Facelets features include the following:**
  - Use of XHTML for creating web pages
  - Support for Facelets tag libraries in addition to JSF and JSTL tag libraries
  - Support for the Expression Language (EL)
    - EL represents a union of the Els offered by JSF and JSP technologies
  - Templating for components and pages
- **Advantages of Facelets for large-scale development projects include the following:**
  - Support for code reuse through templating and composite components
  - Functional extensibility of components and other server-side objects through customization
  - Faster compilation time
  - Compile-time EL validation
  - High-performance rendering

22

Tag Libraries Supported by Facelets

Tag Library	URI	Prefix	Example	Contents
JavaServer Faces Facelets Tag Library	http://java.sun.com/jsf/facelets	ui:	ui:component ui:insert	Tags for templating
JavaServer Faces HTML Tag Library	http://java.sun.com/jsf/html	h:	h:head h:body h:outputText h:inputText	JavaServer Faces component tags for allUIComponent objects
JavaServer Faces Core Tag Library	http://java.sun.com/jsf/core	f:	f:actionListener f:attribute	Tags for JavaServer Faces custom actions that are independent of any particular render kit
JSTL Core Tag Library	http://java.sun.com/jsp/jstl/core	c:	c:forEach c:catch	JSTL 1.2 Core Tags
JSTL Functions Tag Library	http://java.sun.com/jsp/jstl/functions	fn:	fn:toUpperCase fn:toLowerCase	JSTL 1.2 Functions Tags

Java Server Faces (JSF) Expression Language

Expression Language (1)

- **The EL is used by both JSF technology and JSP technology.**
  - The EL represents a union of the expression languages offered by JSF and JSP technologies
- **JSF uses the EL for the following functions:**
  - Deferred and immediate evaluation of expressions
  - The ability to set as well as get data
  - The ability to invoke methods
- **EL provides a way to use simple expressions to perform the following tasks:**
  - Dynamically read application data stored in JavaBeans components, various data structures, and implicit objects
  - Dynamically write data, such as user input into forms, to JavaBeans components
  - Invoke arbitrary static and public methods
  - Dynamically perform arithmetic operations

Expression Language (2)

- The EL is also used to specify the following kinds of expressions that a custom tag attribute will accept:
  - Immediate evaluation expressions or deferred evaluation expressions.
    - An immediate evaluation expression is evaluated at once by the underlying technology, such as JavaServer Faces.
    - A deferred evaluation expression can be evaluated later by the underlying technology using the EL.
  - Value expression or method expression
    - A value expression references data
    - A method expression invokes a method
  - Rvalue expression or lvalue expression
    - An rvalue expression can only read a value, whereas an lvalue expression can both read and write that value to an external object
- EL provides a pluggable API for resolving expressions so custom resolvers that can handle expressions not already supported by the EL can be implemented.

Expression Language (3)

Example

- ${customer.name} // immediate evaluation syntax
- #{customer.name} // deferred evaluation syntax
  - The first expression accesses the name property, gets its value, adds the value to the response, and gets rendered on the page.
  - The same can happen with the second expression. However, the tag handler can defer the evaluation of this expression to a later time in the page lifecycle, if the technology using this tag allows.
- In the case of JSF technology, the second expression is evaluated immediately during an initial request for the page.
  - In this case, this expression acts as an rvalue expression.
  - During a postback request, this expression can be used to set the value of the name property with user input. In this case, the expression acts as an lvalue expression.

Expression Language (4)

Examples

- #{employee.firstName}
  - Call getFirstName on bean named employee. Output it.
- <h:inputText value="#{employee.firstName}"/>
  - When form displayed, call getFirstName, and if non-empty, fill it in as initial value of textfield.
  - When form submitted, validate value and if it is OK, pass value to the setFirstName method
- #{employee.addresses[0].zip}
  - Call getAddresses on bean named employee (which should return an array or list), then take first entry, then call getZip on that, then output it

Testing of the EL: Example

Bean

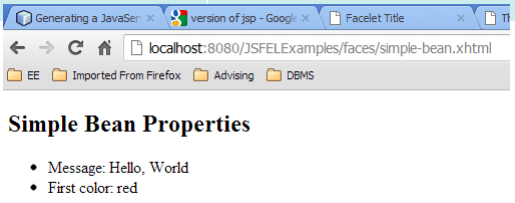
```
@ManagedBean
public class SimpleBean {
  private String[] colors =
    { "red", "orange", "yellow" };

  public String getMessage() {
    return("Hello, World");
  }

  public String[] getColors() {
    return(colors);
  }
}
```

Test Page

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
...
<ul>
  <li>Message: #{simpleBean.message}</li>
  <li>First color: #{simpleBean.colors[0]}</li>
</ul>
...
```



Outputting Simple Bean Properties

- **Format**
  - #{varName.propertyName}
  - <h:outputText value="#{varName.propertyName}" …/>
    - For new JSF 2.0 code, top version is usually used unless you need some other attribute of h:outputText (e.g. "rendered")
- **Interpretation**
  - First, find varName
    - Search for "varName" in all defined scopes, from most specific to most general (request, session, application, in that order for standard Web app scopes). Then look in managed bean defs and instantiate if found.
  - Call getPropertyName and output the result
    - This must be a normal zero-arg accessor method. If boolean, name of method could be *is*PropertyName

Bean Properties Example: Java Code

```
@ManagedBean
@ApplicationScoped

public class TestBean1 {
  private Date creationTime = new Date();
  private String greeting = "Hello";

  public Date getCreationTime() {
    return(creationTime);
  }

  public String getGreeting() {
    return(greeting);
  }

  public double getRandomNumber() {
    return(Math.random());
  }
}
```

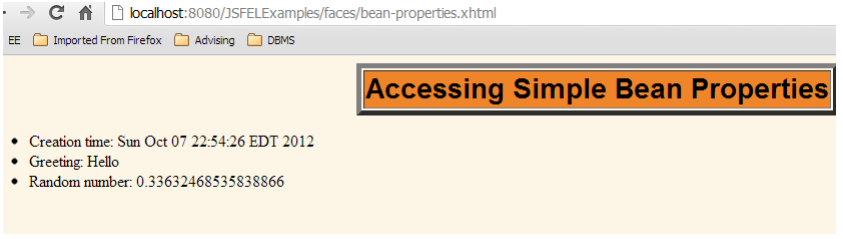
31

Bean Properties Example: Facelets Code

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head><title>Accessing Simple Bean Properties</title>
      <h:outputStylesheet name="css/styles.css"/>
    </h:head>
    <h:body>
      <table border="5" align="center">
        <tr><th class="title">Accessing Simple Bean Properties</th></tr>
      </table>
      <p/>
      <ul>
        <li>Creation time: #{testBean1.creationTime}</li>
        <li>Greeting: #{testBean1.greeting}</li>
        <li>Random number: #{testBean1.randomNumber}</li>
      </ul>
    </h:body>
</html>stBean1.creationTime}
Greeting: #{testBean1.greeting}
Random number: #{testBean1.randomNumber}
```

32

Bean Properties Example: Result



Nested Bean Properties

- **Format**
  - #{var.prop1.prop2.prop3}
  - <h:outputText value="#{var.prop1.prop2.prop3}" …/>
- **Interpretation**
  - First, find var
    - Same as before. Look in existing scopes (narrowest to widest). Use if found. If not found, look in managed bean defs and instantiate.
  - Call getProp1 on bean
  - Call getProp2 on result of getProp1
  - Call getProp3 on result of getProp2
    - And then output the result

Nested Properties Example: Name

```
public class Name {
  private String firstName, lastName;

  public Name(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  public String getFirstName() {
    return(firstName);
  }

  public void setFirstName(String newFirstName) {
    firstName = newFirstName;
  }
  ...
}
```


Nested Properties Example: Company

```
public class Company {
  private String companyName, business;

  public Company(String companyName, String business) {
    this.companyName = companyName;
    this.business = business;
  }

  public String getCompanyName() { return(companyName); }

  public void setCompanyName(String newCompanyName) {
    companyName = newCompanyName;
  }
  ...
}
```

Nested Properties Example: Employee

```
public class Employee {
  private Name name;
  private Company company;

  public Employee(Name name, Company company) {
    this.name = name;
    this.company = company;
  }

  public Name getName() { return(name); }

  public Company getCompany() { return(company); }

  ...
}
```


Nested Properties Example: Employee1

```
@ManagedBean
public class Employee1 extends Employee {
  public Employee1() {
    super(new Name("Bob", "Schneider"),
          new Company("lehman.com",
                      "EE Consulting"));
  }
}
```

Nested Properties Example: Facelets Code

```
...
<ul>
  <li>Employee's first name:
      #{employee1.name.firstName}</li>
  <li>Employee's last name:
      #{employee1.name.lastName}</li>
  <li>Name of employee's company:
      #{employee1.company.companyName}</li>
  <li>Business area of employee's company:
      #{employee1.company.business}</li>
</ul>
...
```


Nested Properties Example: Result

Three Uses of #{...} Expression

- **Designating output value**
  - #{employee.address} or <h:outputText value="#{employee.address}"/>
    - Anytime accessed, means to output getAddress
  - <h:inputText value="#{employee.address}"/>
    - When form initially displayed, means to prepopulate field. Call getAddress and put value in field if non-empty.
- **Designating submitted value**
  - <h:inputText value="#{employee.address}"/>
    - When form submitted, designates where value stored. Pass textfield value to setAddress.
- **Designating method call after submission**
  - <h:commandButton value="Button Label" action="#{employee.processEmployee}"/>
    - When form submitted, designates action handler. This is exact method name, not a shorthand for it.

41

Getter vs. Setter Method Correspondence

- **Example**
  - <h:inputText value="#{myBean.a.b.c.d}"/>
- **When displaying form**
  - Find or instantiate myBean. Call getA. Call getB on result. Call getC on that result. Call getD on that result. If non-empty use as initial value of textfield.
- **When submitting form**
  - Find myBean (instantiate new version if in request scope). Call getA. Call getB on result. Call getC on that result. Then pass submitted value to the setD method of that result.
    - Point: only *final* one becomes setter on submission.
    - This assumes value passes validation.

42

Submitting Properties Example: Employee

```
public class Employee {
  private Name name;
  private Company company;

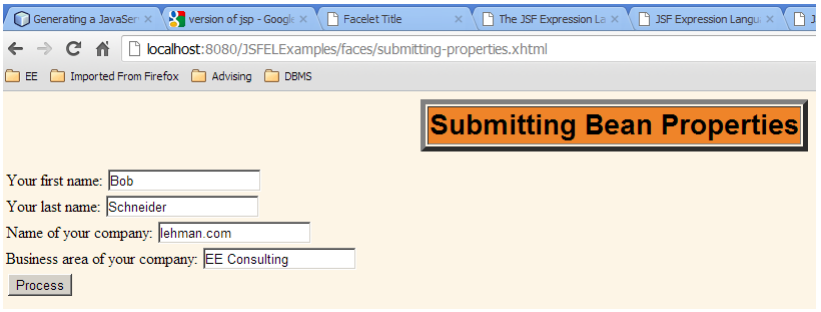
  …

  public String processEmployee() {
    if (Math.random() < 0.5) {
      return("accepted");
    } else {
      return("rejected");
    }
  }
}
```


Submitting Properties Example: Facelet for Form

```
…
<h:form>
Your first name:
<h:inputText value="#{employee1.name.firstName}"/>
<br/>
Your last name:
<h:inputText value="#{employee1.name.lastName}"/>
<br/>
Name of your company:
<h:inputText value="#{employee1.company.companyName}"/>
<br/>
Business area of your company:
<h:inputText value="#{employee1.company.business}"/>
<br/>
<h:commandButton value="Process"
                 action="#{employee1.processEmployee}"/>
</h:form>
```

Submitting Properties Example: Input Page Initial Result



Submitting Properties Example: accepted.xhtml

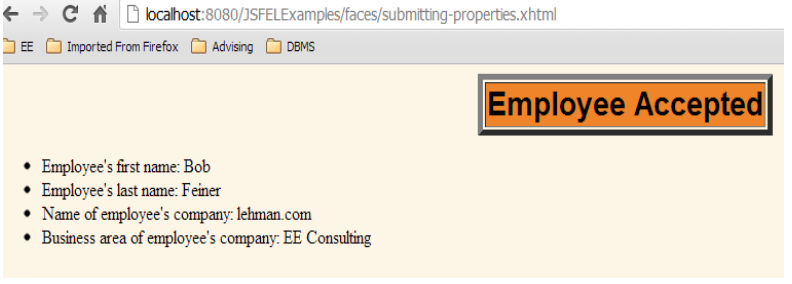
```
…
<table border="5" align="center">
 <tr><th class="title">Employee Accepted</th></tr>
</table>
<p/>
<ul>
 <li>Employee's first name:
    #{employee1.name.firstName}</li>
 <li>Employee's last name:
    #{employee1.name.lastName}</li>
 <li>Name of employee's company:
    #{employee1.company.companyName}</li>
 <li>Business area of employee's company:
    #{employee1.company.business}</li>
</ul>
…
```

Submitting Properties Example: rejected.xhtml

```
...
<table border="5" align="center">
  <tr><th class="title">Employee Rejected</th></tr>
</table>
<p/>
<ul>
  <li>Employee's first name:
     #{employee1.name.firstName}</li>
  <li>Employee's last name:
     #{employee1.name.lastName}</li>
  <li>Name of employee's company:
     #{employee1.company.companyName}</li>
  <li>Business area of employee's company:
     #{employee1.company.business}</li>
</ul>
...
```

47

Submitting Properties Example: Results



48

Equivalence of Dot and Array Notations

- **Equivalent forms**
  - #{name.property}
    - Only legal if "property" would be legal Java variable name
  - #{name["property"]}
- **Reasons for using bracket notation**
  - To access arrays, lists, and other collections
  - To calculate the property name at request time.
    - #{name1[name2]} (no quotes around name2)
  - To use names that are illegal as Java variable names
    - #{foo["bar-baz"]}
    - #{foo["bar.baz"]}

Using the [ ] Form

- **Works for**
  - Array. Equivalent to
    - theArray[index] (getting and setting)
  - List. Equivalent to
    - theList.get(index) or theList.set(index, *submittedVal*)
  - Map. Equivalent to
    - theMap.get(key) or theMap.put(key, *submittedVal*)
- **Equivalent forms (for HashMap)**
  - #{stateCapitals["maryland"]}
  - #{stateCapitals.maryland}

Collections Example: Purchases

```
@ManagedBean
public class Purchases {
  private String[] cheapItems =
    { "Gum", "Yo-yo", "Pencil" };
  private List<String> mediumItems =
    new ArrayList<String>();
  private Map<String,String> valuableItems =
    new HashMap<String,String>();
  private boolean isEverythingOK = true;

  public Purchases() {
    mediumItems.add("iPod");
    mediumItems.add("GameBoy");
    mediumItems.add("Cell Phone");
    valuableItems.put("low", "Lamborghini");
    valuableItems.put("medium", "Yacht");
    valuableItems.put("high", "JSF Training Course");
  }
```


Purchases (Cont'd)

```
  public String[] getCheapItems() {
    return(cheapItems);
  }

  public List<String> getMediumItems() {
    return(mediumItems);
  }

  public Map<String,String> getValuableItems() {
    return(valuableItems);
  }
public String purchaseItems() {
    isEverythingOK = Utils.doBusinessLogic(this);
    isEverythingOK = Utils.doDataAccessLogic(this);
    if (isEverythingOK) {
      return("purchase-success");
    } else {
      return("purchase-failure");
    }
  }
}
```

Collections Example: Utils

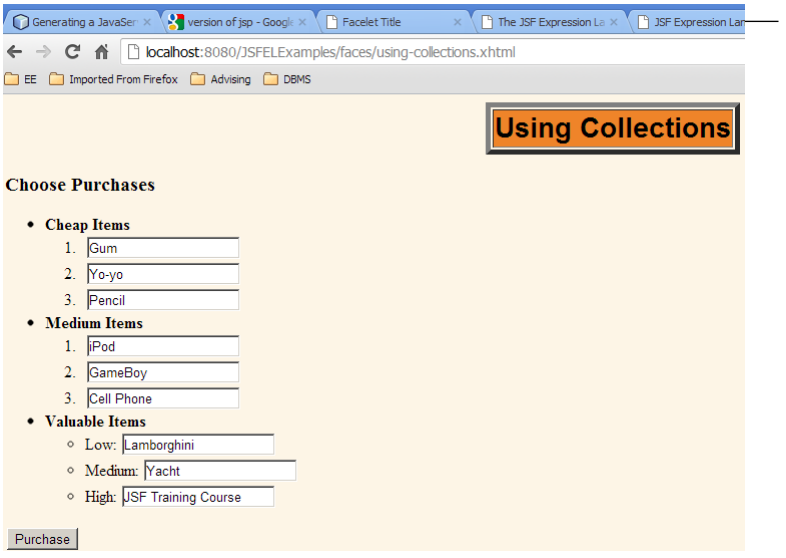
```
public class Utils {
  public static boolean doBusinessLogic
                    (PurchaseBean bean) {
    // Business logic omitted
    return(Math.random() > 0.1);
  }

  public static boolean doDataAccessLogic
                    (PurchaseBean bean) {
    // Database access omitted
    return(Math.random() > 0.1);
  }
}
```


using-collections.xhtml

```
<li><b>Valuable Items</b>
            <ul>
              <li>Low:
                <h:inputText value='#{purchases.valuableItems["low"]}'/>
              </li>
              <li>Medium:
                <h:inputText value='#{purchases.valuableItems["medium"]}'/>
              </li>
              <li>High:
                <h:inputText value='#{purchases.valuableItems["high"]}'/>
              </li>
            </ul></li>
</ul>
        <h:commandButton value="Purchase"
                  action="#{purchases.purchaseItems}"/>
    </h:form>
  </h:body>
```

Input Page Initial Result



55

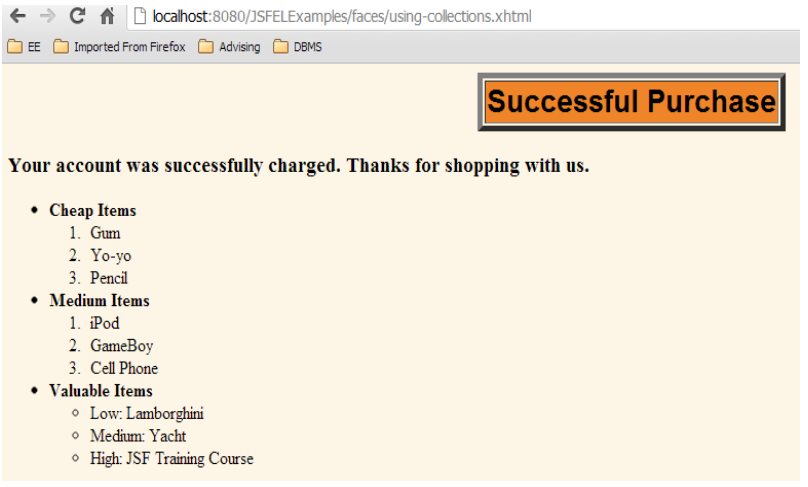
purchase-success.xhtml

purchase-failure.xhtml is very similar.

```
...
<li><b>Cheap Items</b>
<ol>
<li>#{purchases.cheapItems[0]}</li>
<li>#{purchases.cheapItems[1]}</li>
<li>#{purchases.cheapItems[2]}</li>
</ol></li>
<li><b>Medium Items</b>
<ol>
<li>#{purchases.mediumItems[0]}</li>
<li>#{purchases.mediumItems[1]}</li>
<li>#{purchases.mediumItems[2]}</li>
</ol></li>
<li><b>Valuable Items</b>
<ul>
<li>Low: #{purchases.valuableItems["low"]}</li>
<li>Medium: #{purchases.valuableItems["medium"]}</li>
<li>High: #{purchases.valuableItems["high"]}</li>
</ul></li>
</ul>
```

56

Submitting Properties Example: Results



Implicit Objects

- **JSF EL has almost the same predefined implicit objects as JSP EL**
- **Predefined variables**
  - facesContext. The FacesContext object.
    - E.g. #{facesContext.externalContext.remoteUser}
  - param. Request params.
    - E.g. #{param.custID}
  - header. Request headers.
    - E.g. #{header.Accept} or #{header["Accept"]}
    - #{header["Accept-Encoding"]}
  - cookie. Cookie object (not cookie value).
    - E.g. #{cookie.userCookie.value} or #{cookie["userCookie"].value}
  - request, session
    - #{request.queryString}, #{session.id}
  - initParam. Context initialization param.
  - requestScope, sessionScope, applicationScope, etc.
    - Instead of searching scopes.

Example: Implicit Objects (Facelets Code)

```
...
<ul>
  <li><b>Value of JSESSIONID cookie:</b>
      #{cookie.JSESSIONID.value}</li>
  <li><b>The "test" request parameter:</b>
      #{param.test}</li>
  <li><b>User-Agent request header:</b>
      #{header["User-Agent"]}</li>
</ul>
...
```


Example: Implicit Objects (Result)

Expression Language Operators

- **Arithmetic**
  - + - * / div % mod
- **Relational**
  - == eq != ne < lt > gt <= le >= ge
- **Logical**
  - && and || or ! Not
- **Empty**
  - empty
    - True for null, empty string, empty array, empty list, empty map. False otherwise.
- **Notes**
  - Use lt, gt, ge, and, etc. instead of <, >, >=, && (XML)
  - Use operators sparingly to preserve MVC model

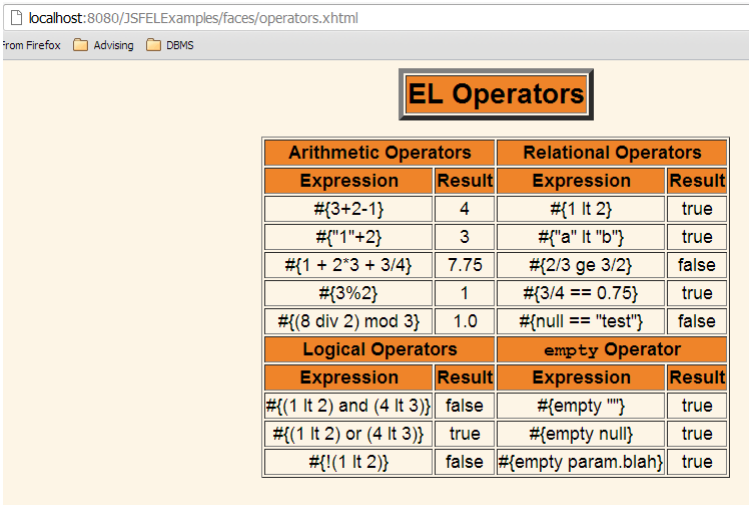
61

Example: Operators

```
…
<table border="1" align="center">
  …
  <tr align="center">
    <td>\#{3+2-1}</td><td>#{3+2-1}</td>
    <td>\#{1 lt 2}</td><td>#{1 lt 2}</td></tr>
  <tr align="center">
    <td>\#{"1"+2}</td><td>#{"1"+2}</td>
    <td>\#{"a" lt "b"}</td><td>#{"a" lt "b"}</td></tr>
  <tr align="center">
    <td>\#{1 + 2*3 + 3/4}</td><td>#{1 + 2*3 + 3/4}</td>
    <td>\#{2/3 ge 3/2}</td><td>#{2/3 ge 3/2}</td></tr>
  <tr align="center">
    <td>\#{3%2}</td><td>#{3%2}</td>
    <td>\#{3/4 == 0.75}</td><td>#{3/4 == 0.75}</td></tr>
  …
</table>
…
```

62

Example: Operators (Result)



Arithmetic Operators		Relational Operators	
Expression	Result	Expression	Result
#{3+2-1}	4	#{1 lt 2}	true
#{"1"+2}	3	#{"a" lt "b"}	true
#{1 + 2*3 + 3/4}	7.75	#{2/3 ge 3/2}	false
#{3%2}	1	#{3/4 == 0.75}	true
#{(8 div 2) mod 3}	1.0	#{null == "test"}	false
Logical Operators		**`empty` Operator**	
Expression	Result	Expression	Result
#{(1 lt 2) and (4 lt 3)}	false	#{empty ""}	true
#{(1 lt 2) or (4 lt 3)}	true	#{empty null}	true
#{!(1 lt 2)}	false	#{empty param.blah}	true

Conditional Text with #{ condition ? val1 : val2 }

- **Idea**
  - The EL directly supports limited conditional output via the ternary operator (test ? thenResult : elseResult). Supply a boolean for the test, put conditional content after the "?" and/or the ":". Values can be literal strings or EL expressions, but they cannot contain HTML tags.
    - Note: you are not permitted to omit the "else" part!
- **Examples**
  - <td class="#{customer.balance < 0 ? 'red': 'black'}">
  - #{ !status.last ? ',' : ''}
- **When used**
  - When you are outputting simple text (no HTML).

Conditional Text with h:outputText and "rendered"

- **Idea**
  - Pass a boolean to the "rendered" attribute, put conditional content in "value" attribute. The value can be a literal string or an EL expression, but the literal string cannot contain HTML tags.
- **Examples**
  - `<h:outputText rendered="#{!status.last}" value=","/>`
  - `<h:outputText rendered="#{status.index > 5}" value="#{user.someWarning}" escape="false"/>`
- **When used**
  - When you are outputting simple text (no HTML) or when the HTML comes from a bean.

Conditional Text with ui:fragment

- **Idea**
  - Pass a boolean to the "rendered" attribute, put conditional content in body content. The value can be a literal string or an EL expression, and the literal string *can* contain HTML tags.
- **Example**
  - ```
    <ui:fragment rendered="#{!status.last}">
        ,
    </ui:fragment>
    ```
- **When used**
  - When you are outputting literal HTML.
    - Can always be used in lieu of h:outputText, but if no HTML, h:outputText is more succinct.
    ```

Passing Arguments to Methods

- **EL version 2.1 lets you call regular methods**
  - Rather than only zero-arg accessor methods
- **Syntax**
  - Basic syntax is straightforward
    - #{someBean.someMethod(arg1, arg2)}
  - The arguments can also be EL expressions
- **Cautions**
  - Use sparingly: put complexity in Java, not facelets
  - Works only in Java EE 6 App Servers.
    - E.g., works in Glassfish 3.x, JBoss 6, and WebLogic11g.

Method Args: Java Code

```
@ManagedBean
@ApplicationScoped
public class TestBean2 {
  private final String HELLO_ENGLISH = "Hello!";
  private final String HELLO_SPANISH = "¡Hola!";

  public String greeting(boolean useSpanish) {
    if (useSpanish) {
      return(HELLO_SPANISH);
    } else {
      return(HELLO_ENGLISH);
    }
  }

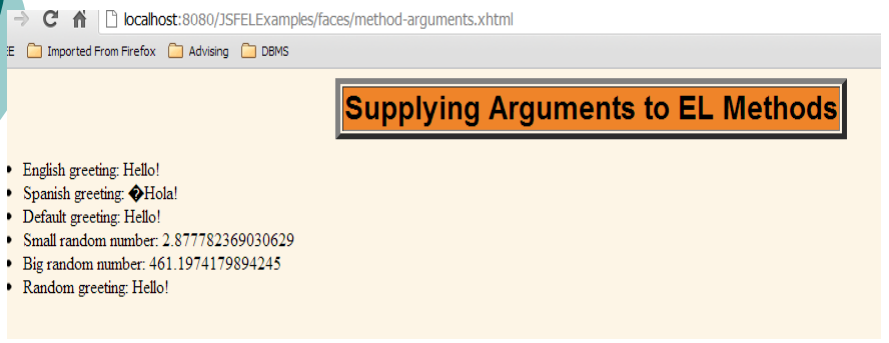
  public String greeting() {
    return(greeting(false));
  }

  public double randomNumber(double range) {
    return(range * Math.random());
  }
```

Method Args: Facelets Code

```
...
<ul>
  <li>English greeting: #{testBean2.greeting(false)}</li>
  <li>Spanish greeting: #{testBean2.greeting(true)}</li>
  <li>Default greeting: #{testBean2.greeting()}</li>
  <li>Small random number: #{testBean2.randomNumber(5)}</li>
  <li>Big random number: #{testBean2.randomNumber(500)}</li>
  <li>Random greeting:
      #{testBean2.greeting(testBean2.randomNumber(200) gt 100)}
  </li>
</ul>
...
```

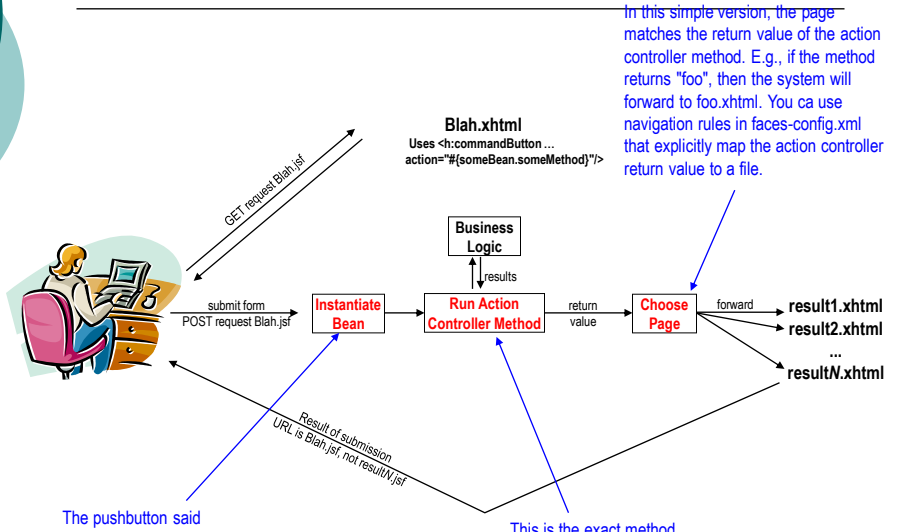

Method Args: Results



More on JSF Managed Beans

71

JSF Flow of Control (Simplified)



In this simple version, the page matches the return value of the action controller method. E.g., if the method returns "foo", then the system will forward to foo.xhtml. You ca use navigation rules in faces-config.xml that explicitly map the action controller return value to a file.

The pushbutton said action="#{someBean.someMethod}". So, instantiate bean whose name is someBean.

This is the exact method name given in the pushbutton's action.

72

Main Points

- **@ManagedBean annotation**

  ```
  @ManagedBean
  public class SomeName { … }
  ```

  - You refer to bean with #{someName.blah}, where bean name is class name (minus packages) with first letter changed to lower case. Request scoped by default.
    - And "blah" is either an exact method name (as with action of h:commandButton), or a shortcut for a getter and setter method (as with value of h:inputText).
- **Return values of action controller method**
  - If action controller method returns "foo" and "bar" and there are no explicit mappings in faces-config.xml, then results pages are foo.xhtml and bar.xhtml
    - From same folder that contained the form

Example

- **Idea**
  - Click on button in initial page
  - Get one of three results pages, chosen at random
- **What you need**
  - A starting page
    - <h:commandButton…action="#{simpleBean.doNavigation}"/>
  - A bean
    - Class: SimpleBean (bean name above except for case)
    - @ManagedBean annotation
    - doNavigation method returns 3 possible Strings
      - "page1", "page2", or "page3"
  - Three results pages
    - Names match return values of doNavigation method
      - page1.xhtml, page2.xhtml, and page3.xhtml

Basic Beans

- **Java classes that follow certain conventions**
  - Must have a zero-argument (empty) constructor
    - You can satisfy this requirement either by explicitly defining such a constructor or by omitting all constructors
  - Should have no public instance variables (fields)
    - You should already follow this practice and use accessor methods instead of allowing direct access to fields
  - Persistent values should be accessed through methods called get*Blah* and set*Blah*
    - If class has method getTitle that returns a String, class is said to have a String *property* named title
      - JSF uses #{book.title} to mean "call getTitle on 'book' ".
    - Boolean properties may use is*Blah* instead of get*Blah*
    - What matters is method name, not instance variable name

More on Bean Properties

- **Usual rule to turn method into property**
  - Drop the word "get" or "set" and change the next letter to lowercase. Again, instance var name is irrelevant.
    - Method name: getFirstName
    - Property name: firstName
    - Example: #{customer.firstName}
- **Exception 1: boolean properties**
  - If getter returns boolean or Boolean
    - Method name: getPrime or isPrime
    - Property name: prime
    - Example: #{myNumber.prime}
- **Exception 2: consecutive uppercase letters**
  - If two uppercase letters in a row after "get" or "set"
    - Method name: getURL
    - Property name: URL (not uRL)
    - Example: #{webSite.URL}

Bean Properties: Examples

Method Names	Property Name	Example JSF Usage
getFirstName setFirstName	firstName	#{customer.firstName} <h:inputText value="#{customer.firstName}"/>
isExecutive setExecutive (boolean property)	executive	#{customer.executive} <h:selectBooleanCheckbox value="#{customer.executive}"/>
getExecutive setExecutive (boolean property)	executive	#{customer.executive} <h:selectBooleanCheckbox value="#{customer.executive}"/>
getZIP setZIP	ZIP	#{address.ZIP} <h:inputText value="#{address.ZIP}"/>

Note 1: property name does not exist anywhere in your code. It is just a shortcut for the method name.
Note 2: property name is derived only from method name. Instance variable name is irrelevant.

77

Why You Should Use Accessors, Not Public Fields

- **Bean rules**
  - To be a bean, you should not have public fields.
  - Wrong

```
public double speed;
```

  - Right

```
private double speed;  // Var need not match method name

public double getSpeed() {
    return(speed);
}
public void setSpeed(double speed) {
    this.speed = speed;
}
```

- **OOP design**
  - You should do this in all your Java code anyhow. Why?

78

Why You Should Use Accessors, Not Public Fields

1) You can put constraints on values

```
public void setSpeed(double newSpeed) {
  if (newSpeed < 0) {
    sendErrorMessage(...);
    newSpeed = Math.abs(newSpeed);
  }
  speed = newSpeed;
}
```

- If users of your class accessed the fields directly, then they would each be responsible for checking constraints.

79

Why You Should Use Accessors, Not Public Fields

2) You can change your internal representation without changing interface

```
// Now using metric units (kph, not mph)

public void setSpeed(double newSpeed) {
  speedInKPH = convert(newSpeed);
}

public void setSpeedInKPH(double newSpeed) {
  speedInKPH = newSpeed;
}
```

80

Why You Should Use Accessors, Not Public Fields

- **3) You can perform arbitrary side effects**

```
public double setSpeed(double newSpeed) {
   speed = newSpeed;
   updateSpeedometerDisplay();
}
```

  - If users of your class accessed the fields directly, then they would each be responsible for executing side effects. Too much work and runs huge risk of having display inconsistent from actual values.

Basic Beans: Bottom Line

- **It is no onerous requirement to be a "bean"**
  - You are probably following most of the conventions already anyhow
    - Zero arg constructor
    - No public instance variables
    - Use getBlah/setBlah or isBlah/setBlah naming conventions
- **JSF often refers to "bean properties"**
  - Which are shortcuts for getter/setter methods
    - getFirstName method: refer to "firstName"
      - #{customer.firstName}
    - isVeryCool method (boolean): refer to "veryCool"
      - #{invention.veryCool}
    - getHTMLString method: refer to "HTMLString"
      - #{message.HTMLString}

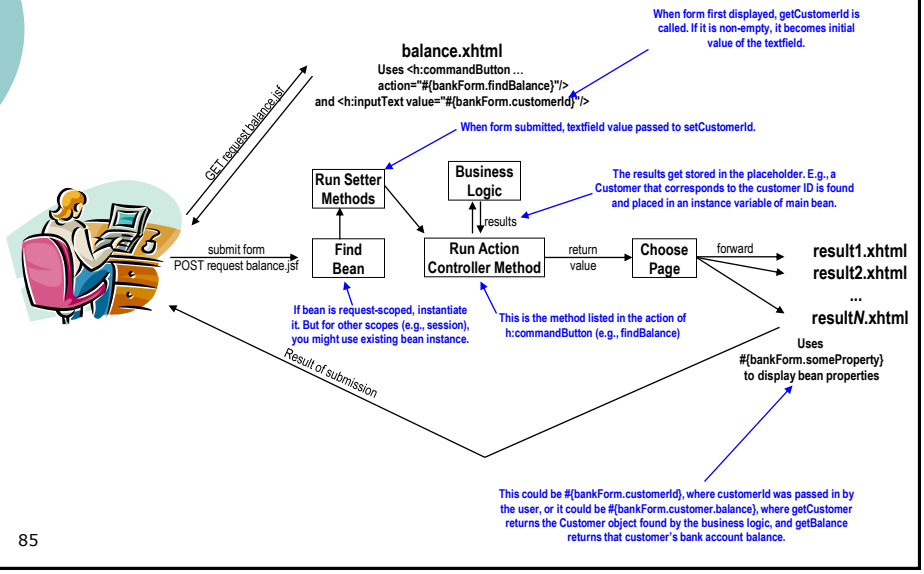
Managed Beans

- **JSF automatically "manages" the bean**
  - Instantiates it
    - Thus the need for a zero-arg constructor
  - Controls its lifecycle
    - Scope (request, session, application) determines lifetime
  - Calls setter methods
    - I.e., for <h:inputText value="#{customer.firstName"/>, when form submitted, the value is passed to setFirstName
  - Calls getter methods
    - #{customer.firstName} results in calling getFirstName
- **Declaring beans**
  - Simplest: @ManagedBean before class

Three Parts of the Managed Bean

- **Managed beans typically have three parts**
  - Bean properties (i.e, pairs of getter and setter methods)
    - One pair for each input element
    - Setter methods called automatically by JSF when form submitted
  - Action controller methods
    - Often only one, but could be several if the same form has multiple push buttons
    - Action controller method (corresponding to the button that was pressed) called automatically by JSF
  - Placeholders for results data
    - Not automatically called by JSF: to be filled in by action controller method based on results of business logic.

JSF Flow of Control



Example

- **Idea**
  - Enter a bank customer ID
  - Get either
    - Page showing first name, last name, and balance
    - Error message about invalid customer ID
- **What managed bean needs**
  - Bean property corresponding to input element
    - I.e., getCustomerId and setCustomerId
  - Action controller method
    - Maps a customer ID to a Customer and stores the Customer in an instance variable
  - Placeholder for results data
    - An initially empty instance variable (for storing the Customer) and associated getter method.

Input Form (customer-lookup.xhtml)

```
<!DOCTYPE …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
…
<h:body>
…
<fieldset>
<legend>Enter Your Customer ID</legend>
<p>Legal ids are id001, id002, and id003.</p>
<h:form>
  Customer ID:
  <h:inputText value="#{bankForm.customerId}"/><br/>
  <h:commandButton value="Show Current Balance"
                   action="#{bankForm.findBalance}"/>
</h:form>
</fieldset>
…
</h:body></html>
```


BankForm.java: Part 1 (Bean Properties for Input Elements)

```
@ManagedBean
public class BankForm {
  private String customerId;

  public String getCustomerId() {
    return(customerId);
  }

  public void setCustomerId(String customerId) {
    this.customerId = customerId;
  }
```

Called by JSF when form first displayed. Since it returns null in that case, the textfield is left blank initially.

When form submitted, the bean is instantiated again (since it is request-scoped, not session-scoped) and the value in the textfield is passed to this method.

BankForm.java: Part 2 (Action Controller Method)

```
private static CustomerLookupService lookupService =
  new CustomerSimpleMap();

public String findBalance() {
  customer = lookupService.findCustomer(customerId);
  if (customer == null) {
    return("unknown-customer");
  } else {
    return("show-customer");
  }
}
```

This is the placeholder

Filled in by JSF before this action controller method is called.

89

BankForm.java: Part 3 (Placeholder for Results)

```
private Customer customer;

public Customer getCustomer() {
  return(customer);
}
}
```

Filled in by the action controller method based on the value returned by the business logic.

The getCustomer method is needed because the results page does #{bankForm.customer.firstName} and #{bankForm.customer.otherProperties}. But no setter method is needed since this property does not correspond directly to input data, and this property is not automatically filled in by JSF.

90

Business Logic (Interface)

```
public interface CustomerLookupService {
  public Customer findCustomer(String id);
}
```

91

Business Logic (Implementation)

```
public class CustomerSimpleMap
      implements CustomerLookupService {
  private Map<String,Customer> customers;

  public CustomerSimpleMap() {
    customers = new HashMap<String,Customer>();
    addCustomer(new Customer("id001", "Harry",
                             "Hacker", -3456.78));
    addCustomer(new Customer("id002", "Codie",
                             "Coder", 1234.56));
    addCustomer(new Customer("id003", "Polly",
                             "Programmer", 987654.32));
  }
```

92

Business Logic (Implementation, Continued)

```
  public Customer findCustomer(String id) {
    if (id!=null) {
      return(customers.get(id.toLowerCase()));
    } else {
      return(null);
    }
  }

  private void addCustomer(Customer customer) {
    customers.put(customer.getId(), customer);
  }
}
```

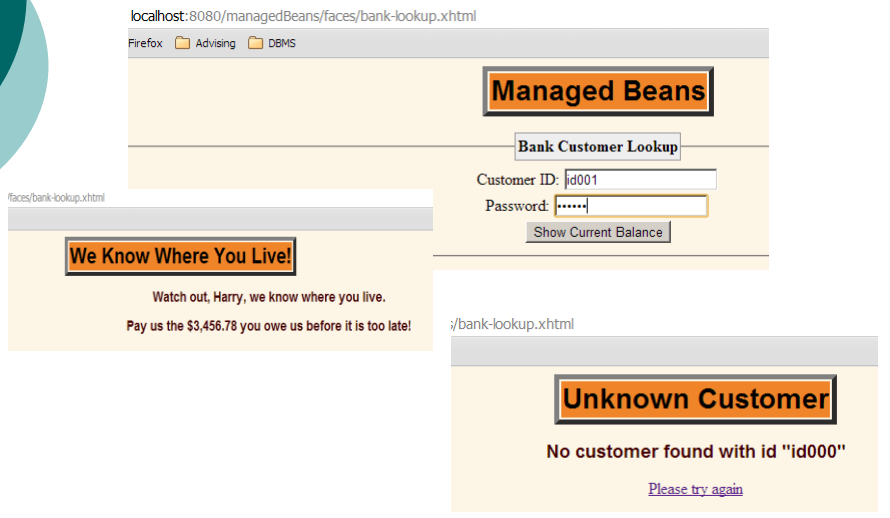

Results Page 1 (show-balance.xhtml)

```
<!DOCTYPE …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
<h:head>
…
</h:head>
<h:body>
…
<ul>
  <li>First name: #{bankForm.customer.firstName}</li>
  <li>Last name: #{bankForm.customer.lastName}</li>
  <li>ID: #{bankForm.customer.id}</li>
  <li>Balance: $#{bankForm.customer.balanceNoSign}</li>
</ul>
…
</h:body></html>
```

Results Page 2 (unknown-customer.xhtml)

```
<!DOCTYPE …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
<h:head>
…
</h:head>
<h:body>
…
<h2>No customer found with id "#{bankForm.customerId}"</h2>
<p>Please <a href="customer-lookup.jsf">try again</a>.</p>
…
</h:body></html>
```


Results



Bean Property

- **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 (not null or ""), this is initial value inside textfield. Not used for password fields.
  - <h:selectBooleanCheckbox…/> (checkbox)
    - Getter returns true: initially checked. Otherwise unchecked
  - <h:selectOneMenu…/> (combobox; drop down menu)
    - If return value of getter matches an entry in menu, that is initial selection. Otherwise top entry is initially selected.

Textfields

- **Example 1**
  - <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 (Chaining)**
  - <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.

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)

Drop Down Menus (Comboboxes)

Example

- <h:selectOneMenu
  value="#{someBean.someProperty}">
  <f:selectItems value="#{someBean.choices}"/>
  </h:selectOneMenu>
  - When form initially displayed, call getChoices (which should return List<SelectItem> or SelectItem[]). 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.
  - There are several forms of the SelectItem constructor, but the simplest just takes a String: the value that will go in the menu.
  - When form submitted, pass the current selection to setSomeProperty.
  - Note that, due to the f:selectItems tag, you must add an entry for the f: namespace to the html start tag at top of file

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.

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 Balmer, and James Gosling?
  <h:selectBooleanCheckbox value="#{trainingForm.liar}"/><br/>
  <h:commandButton value="Show Recommended Study Plan"
                   action="#{trainingForm.showTrainingPlan}"/>
</h:form>
…
</h:body></html>
```

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 List<SelectItem> getAvailableLanguages() {
    return(LanguageUtils.languageList());
  } // Options for dropdown box. See later slide.
```


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.

Managed Bean (Part 3 – Placeholder for Results)

```
private List<String> languagesToStudy;

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


Helper Class (Code for Options in Dropdown)

```
public class LanguageUtils {
  private static String[] languages =
    { "Java", "JavaScript", "C#", "C++", "PHP", "Python", "Perl", "Ruby", "Scala" };
  private static List<SelectItem> availableLanguages;

  static {
    availableLanguages = new ArrayList<SelectItem>();
    for(String language: languages) {
      availableLanguages.add(new SelectItem(language));
    }
  }

  public static List<SelectItem> languageList() {
    return(availableLanguages);
  }
...
```

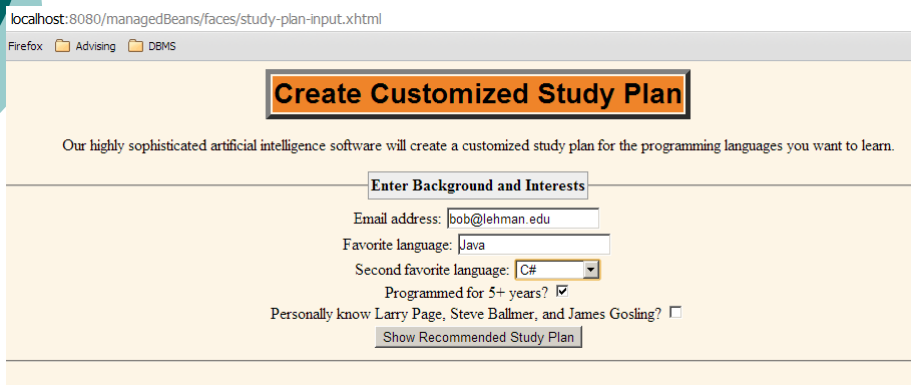
Main Results Page (study-plan.xhtml)

```
<!DOCTYPE …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/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>
```

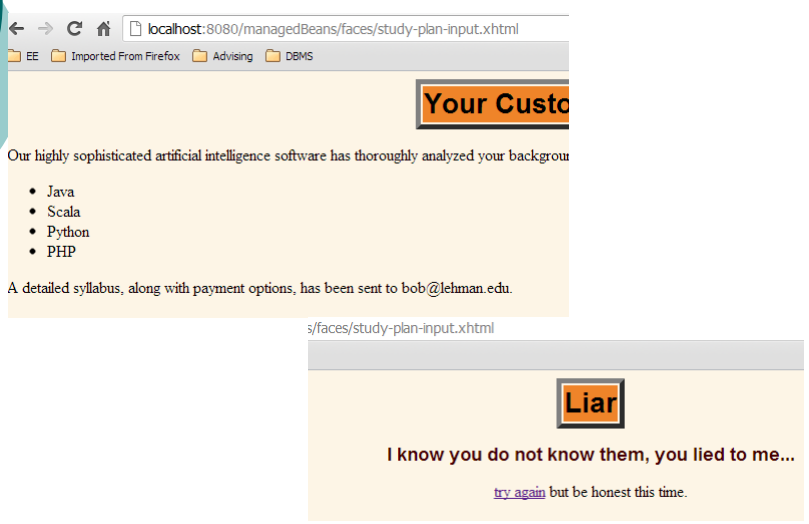

"Error" Page (liar.xhtml)

```
<!DOCTYPE …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
<h:head>…</h:head>
<h:body>
…
<h2>I know you do not know them, you lied to me...</h2>
    <p><h:link outcome="study-plan-input" value="try again"/>
               but be honest this time.</p>…
</h:body></html>
```

Results (Input Form)



Results (Results Pages)



Problem: Getting request & response objects

- **No automatic access to request & response**
  - JSF action controller methods do not have direct access
- **Why this matters**
  - Uses for request object
    - Explicit session manipulation
      - E.g., changing inactive interval or invalidating session
    - Explicit cookie manipulation (e.g., long-lived cookies)
    - Reading request headers (e.g., User-Agent)
    - Looking up requesting host name
  - Uses for response object
    - Setting status codes
    - Setting response headers
    - Setting long-lived cookies

Solution

- **Static methods**
  - If they are needed, use static method calls to get them

```
ExternalContext context =
  FacesContext.getCurrentInstance().getExternalContext();
HttpServletRequest request =
  (HttpServletRequest)context.getRequest();
HttpServletResponse response =
  (HttpServletResponse)context.getResponse();
```

Example

- **Idea**
  - Collect a search string and a search engine name, show the results of a search with that search engine.
- **Input form**
  - Use textfield for arbitrary search string. Use drop down menu to list only search engines that app supports.
- **Managed bean**
  - Construct a URL by concatenating a base URL (e.g., http://www.google.com/search?q=) with the URL-encoded search string
  - Do response.sendRedirect
    - Must use static methods to get the HttpServletResponse
  - Return normal strings for error pages

Input Form (search-engine-form.xhtml)

```
<!DOCTYPE …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
<h:head>…</h:head>
<h:body>
…
<h:form>
  Search String:
  <h:inputText value="#{searchController.searchString}"/><br/>
  Search Engine:
  <h:selectOneMenu value="#{searchController.searchEngine}">
    <f:selectItems value="#{searchController.searchEngineNames}"/>
  </h:selectOneMenu><br/>
  <h:commandButton value="Search"
                   action="#{searchController.doSearch}"/>
</h:form>
</h:body></html>
```

Managed Bean (Part 1 – Properties for Input Elements)

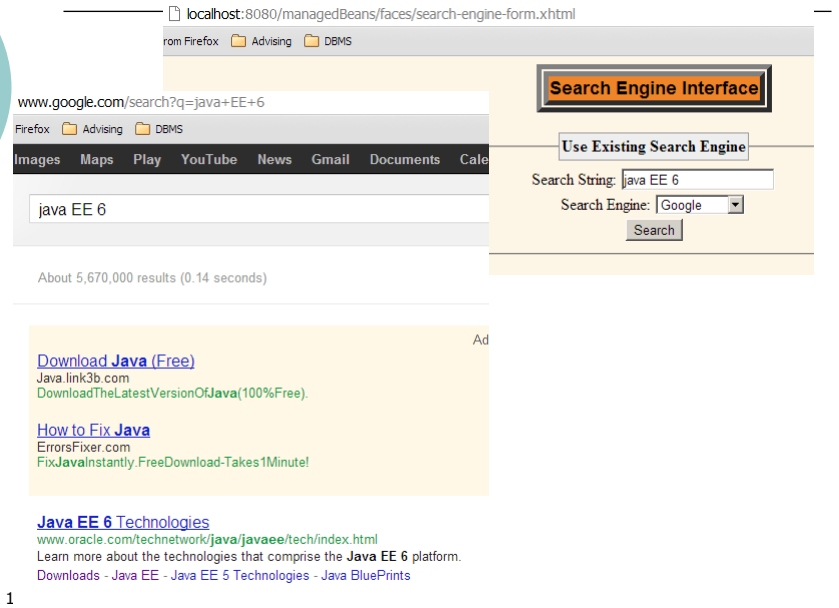
```
@ManagedBean
public class SearchController {
  private String searchString, searchEngine;

  public String getSearchString() {
    return(searchString);
  }
  public void setSearchString(String searchString) {
    this.searchString = searchString;
  }
  public String getSearchEngine() {
    return(searchEngine);
  }
  public void setSearchEngine(String searchEngine) {
    this.searchEngine = searchEngine;
  }
  public List<SelectItem> getSearchEngineNames() {
    return(SearchUtilities.searchEngineNames());
  }
```


Managed Bean (Part 2 – Action Controller)

```
  public String doSearch() throws IOException {
    if ((searchString.trim().length() == 0)) {
      return("no-search-string");
    }
    searchString = URLEncoder.encode(searchString, "utf-8");
    String searchURL =
      SearchUtilities.makeURL(searchEngine, searchString);
    if (searchURL != null) {
      ExternalContext context =
        FacesContext.getCurrentInstance().getExternalContext();
      HttpServletResponse response =
        (HttpServletResponse)context.getResponse();
      response.sendRedirect(searchURL);
      return(null);
    } else {
      return("unknown-search-engine");
    }
  }
```

Results (Input Form)



1

Bean Scopes

- **Idea**
  - Designates how long managed beans will stay "alive", and which users and requests can access previous bean instances.
- **JSF 2.x scopes**
  - request, session, application, view, none, custom
  - Specified either in faces-config.xml or by one of the new annotations (e.g., @SessionScoped)
  - Request scope is still the default

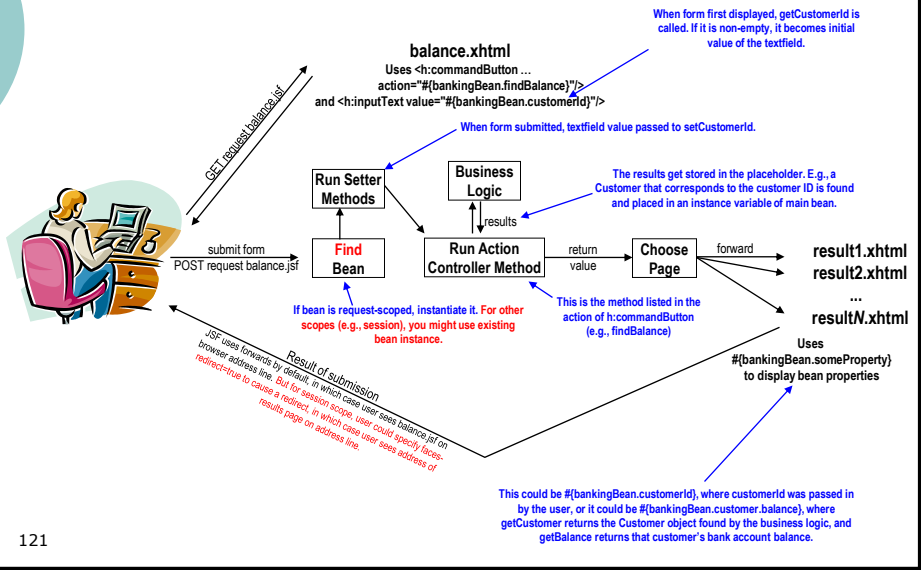
Annotations to Specify Bean Scope (1)

- **@RequestScoped**
  - Default. Make a new instance for every HTTP request. Since beans are also used for initial values in input form, this means bean is generally instantiated twice (once when form is displayed, once when form is submitted).
- **@SessionScoped**
  - Put bean in session scope. If same user with same cookie returns before session timeout, same bean instance is used. You should make bean Serializable.
- **@ApplicationScoped**
  - Put bean in application scope. Shared by all users. Bean either should have no mutable state or you must carefully synchronize access.

Annotations to Specify Bean Scope (2)

- **@ViewScoped**
  - Same bean instance is used as long as same user is on same page (e.g., with event handlers or Ajax).
    - New scope in JSF 2.0.
    - Bean should implement Serializable
- **@CustomScoped(value="#{someMap}")**
  - Bean is stored in the Map, and programmer can control lifecycle.
    - New scope in JSF 2.0
- **@NoneScoped**
  - Bean is not placed in a scope. Useful for beans that are referenced by other beans that *are* in scopes.
    - New scope in JSF 2.0

JSF Flow of Control (Updated)



Main Points

- **You can use annotations to give scope**
  - @RequestScoped (same as omitting scope)
  - @SessionScoped
  - @ApplicationScoped
  - @ViewScoped
  - @CustomScoped
  - @NoneScoped
  - Traditionally placed after @ManagedBean

```
@ManagedBean
@SessionScoped
public class SomePojo { … }
```

Application Scope: When Used

- **For beans that have no changeable state**
  - Navigation rules only (no getters/setters)
    - As in Navigator example earlier (this is rare in real life)
  - Supplies list of choices for drop down menus

```
<h:selectOneMenu value="#{requestScopedBean.choice}">
    <f:selectItems value="#{appScopedBean.options}"/>
</h:selectOneMenu>
```

  - Data structures used as properties in main beans
    - See later section on @ManagedBeanProperty
- **For beans you deliberately share**
  - Shared across all users and all pages
  - You have to very carefully use synchronization to avoid race conditions
    - Quite rarely used

123

Application Scope: Syntax

- **Usual**

```
@ManagedBean
@ApplicationScoped
public class SomeClassNoUserState { … }
```

    - Class is instantiated first time it is used. After that, all users and all requests share the same instance.
- **Occasional**

```
@ManagedBean(eager=true)
@ApplicationScoped
public class SomeClassBigDataNoUserState { … }
```

    - Class is instantiated when the app is loaded (which is usually when the server starts). After that, all users and all requests share the same instance. Useful if the class has a big data structure that takes a long time to initialize.

124

Example: Bean with No Changeable State

- **Simple Page Navigator**
- **There was no state (no instance variables), just an action controller method**
  - So, application scope prevents unnecessary object instantiation.
    - In practice, instantiation of small objects is very fast, so this "optimization" is probably not even measurable.
      - And causes problems if you later go and add input fields and corresponding accessor methods. So, for Navigator example, the benefit of application scope is questionable at best.
    - For large data structures (especially large Maps), this trick can help significantly, though.
      - Large list of choices for drop down menus
      - Maps for business logic
        - In both cases above, previous examples used static variables to work around this problem. Application scope is a reasonable alternative.

Navigator.java

```
package coreservlets;

import javax.faces.bean.*;

@ManagedBean
@ApplicationScoped
public class Navigator {
  private String[] resultPages =
    { "page1", "page2", "page3" };

  public String choosePage() {
    return(RandomUtils.randomElement(resultPages));
  }
}
```

Since there is no mutable state, all users can share the same instance of this bean. For this example, using application scope only saves recreating the tiny array, so has little-to-no measurable performance benefit, and it causes huge problems later if you add accessor methods for storing user-specific data, but forget to remove @ApplicationScoped. Still, if the data structure were large and you had no mutable state, this trick would buy you something.

Session Scope: Main Points

- **Bean instance reused if**
  - Same user
  - Same browser session
    - Usually based on cookies, but can be based on URL rewriting
- **Useful for**
  - Remembering user preferences
  - Prefilling values from previous entries
  - Accumulating lists of user data (ala shopping carts)
- **Normal Java session tracking rules apply**
  - Custom classes should be Serializable
    - Some servers save session data to disk on restart
    - Distributed Web apps need this to replicate sessions

Session Scope: Example

- **Idea**
  - Small variation of banking example
  - Remember previously entered id
    - If end user comes back to input page, the ID they entered last time is already filled in
- **What you need**
  - Add @SessionScoped for bean
  - Make bean Serializable
  - Optionally, add faces-redirect=true to end of return values, to tell JSF to redirect (instead of forward) to results pages
    - Allows users to access results pages directly

bank-lookup2.xhtml

```
<!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:body>
…
<fieldset>
<legend>Bank Customer Lookup (Session Scope)</legend>
<h:form>
  Customer ID:
  <h:inputText value="#{bankingBean2.customerId}"/><br/>
  Password:
  <h:inputSecret value="#{bankingBean2.password}"/><br/>
  <h:commandButton value="Show Current Balance"
                   action="#{bankingBean2.showBalance}"/>
</h:form>
</fieldset>
…
</h:body></html>
```


BankingBean2.java

```
import java.io.*;
import javax.faces.bean.*;

@ManagedBean
@SessionScoped
public class BankingBean2 extends BankingBean
                          implements Serializable {
  @Override
  public String showBalance() {
    String origResult = super.showBalance();
    return(origResult + "2?faces-redirect=true");
  }
}
```

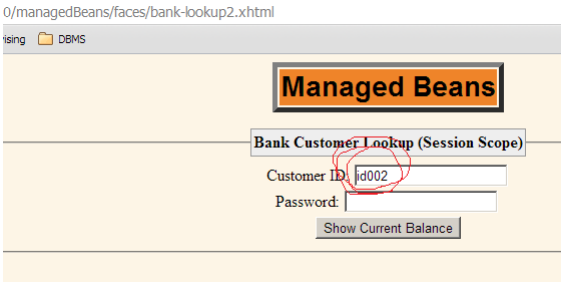
If a page uses the name bankingBean2 and is accessed by the same user in the same browser session, the same bean instance will be used.

Results pages are negative-balance2.xhtml, normal-balance2.xhtml, etc. By also appending faces-redirect=true, JSF will redirect instead of forward to the results pages, thus exposing the URLs of the results pages and letting users navigate directly to them later.

normal-balance2.xhtml

```
<!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>
…
</h:head>
<h:body>
…
<ul>
  <li>First name: #{bankingBean2.customer.firstName}</li>
  <li>Last name: #{bankingBean2.customer.lastName}</li>
  <li>ID: #{bankingBean2.customer.id}</li>
  <li>Balance: $#{bankingBean2.customer.balanceNoSign}</li>
</ul>
…
</h:body></html>
```


Results (User Returns in Same Browser Session)



@ManagedProperty: Main Points

- **JSF supports simple dependency injection**
  - That is, you can assign values to a managed bean property (i.e., a value the main bean depends on) without hardcoding it into the class definition
    - Not as powerful as with Spring, but still useful
  - @ManagedProperty lets you do this with annotations
    - @ManagedProperty(value="#{someBean}")
      private SomeType someField;
  - also <managed-property> lets you do it in faces-config.xml
- **Setter method for the property is required**
  - E.g., in example above, you must have setSomeField
    - You can use "name" attribute of @ManagedProperty if setter method name does not match field name

@ManagedProperty: Secondary Points

- **You can instantiate beans at app load time**
  - @ManagedBean(eager=true)
    @ApplicationScoped
    public class SomePojo { … }
    - This is useful for the bean that you inject into the main bean. The injected bean is often something shared like a lookup service, whereas the main bean usually contains user-specific data
- **Unclear that annotation approach is better**
  - One of the points of dependency injection is to let you change the concrete class without changing any Java code
    - This partially violates that principle
  - faces-config.xml lets you specify Map elements
    - The annotation does not

@ManagedProperty: Example

- **Idea**
  - Refactoring of banking example
  - Customer lookup service will be injected into main bean via @ManagedProperty
  - Lookup service will be application scoped and created at application load time
- **What you need**
  - Main bean

```
@ManagedProperty(value="#{lookupServiceBeanName}")
private CustomerLookupService service;
public void setService(…) { … }
```

  - Lookup service bean (the bean being injected)
    - @ManagedBean(eager=true)
      @ApplicationScoped

bank-lookup3.xhtml

```
<!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:body>
…
<fieldset>
<legend>Bank Customer Lookup</legend>
<h:form>
  Customer ID:
  <h:inputText value="#{bankingBean3.customerId}"/><br/>
  Password:
  <h:inputSecret value="#{bankingBean3.password}"/><br/>
  <h:commandButton value="Show Current Balance"
                   action="#{bankingBean3.showBalance}"/>
</h:form>
</fieldset>
…
</h:body></html>
```

BankingBean3.java

```
@ManagedBean
public class BankingBean3 extends BankingBeanBase {
 @ManagedProperty(value="#{customerSimpleMap2}")
 private CustomerLookupService service;

 public void setService(CustomerLookupService service) {
   this.service = service;
 }
 public String showBalance() {
   if (!password.equals("secret")) {
     return("wrong-password3");
   }
   customer = service.findCustomer(customerId);
   if (customer == null) {
     …
 }
}
```


CustomerSimpleMap2.java

```
@ManagedBean(eager=true)
@ApplicationScoped
public class CustomerSimpleMap2
     extends CustomerSimpleMap {
}
```

normal-balance3.xhtml

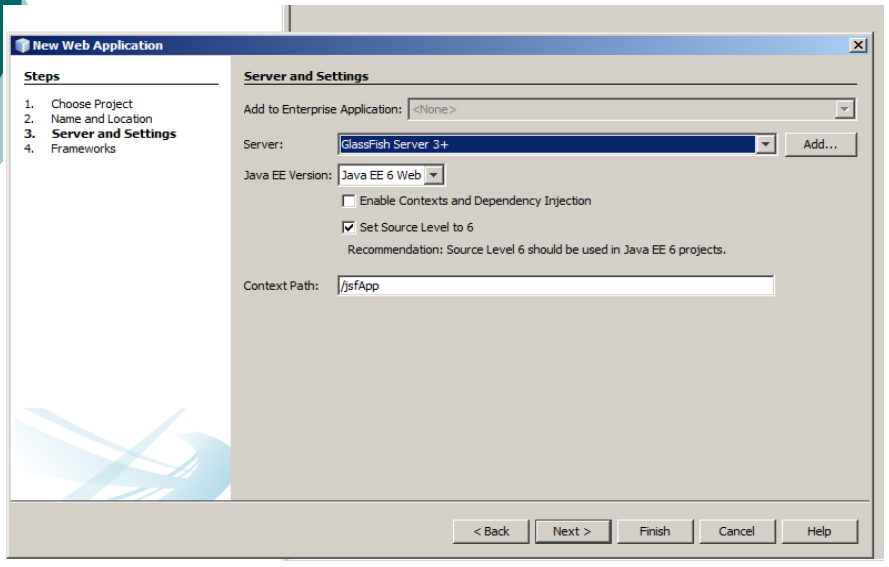
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
<h:head>
…
</h:head>
<h:body>
…
<ul>
  <li>First name: #{bankingBean3.customer.firstName}</li>
  <li>Last name: #{bankingBean3.customer.lastName}</li>
  <li>ID: #{bankingBean3.customer.id}</li>
  <li>Balance: $#{bankingBean3.customer.balanceNoSign}</li>
</ul>
…
</h:body></html>
```


Result

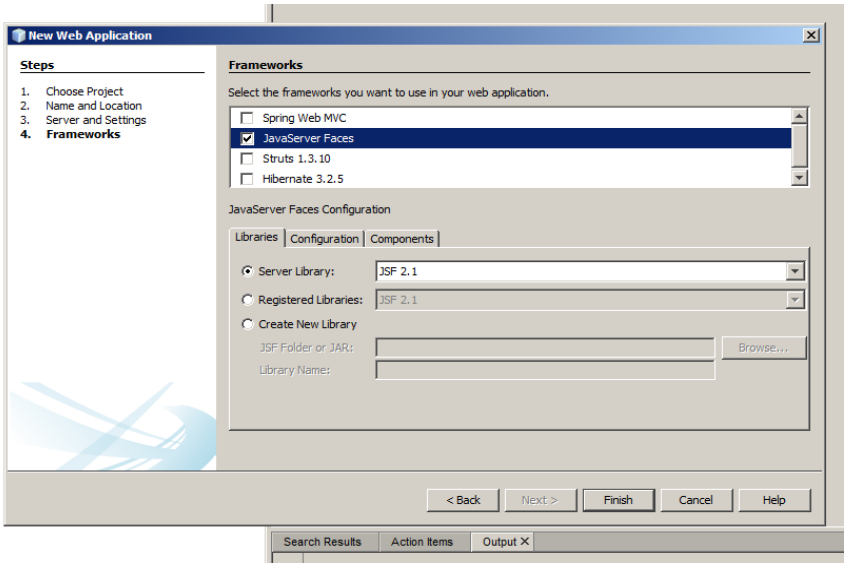
- **Same as the first banking example**

Developing JSF Projects
by NetBeans IDE

Creating a New JSF Project (jsfApp)



Select JSF Framework



Generated Files by the JSF Wizard

- **A single facelet file index.xhtml and web.xml configuration file are created**
  - Servlet 3.0 API does not need web.xml
  - Configuration options can be specified via annotations
  - For JSF applications, it is good idea to have web.xml, since it is used to specify the JSF project stage

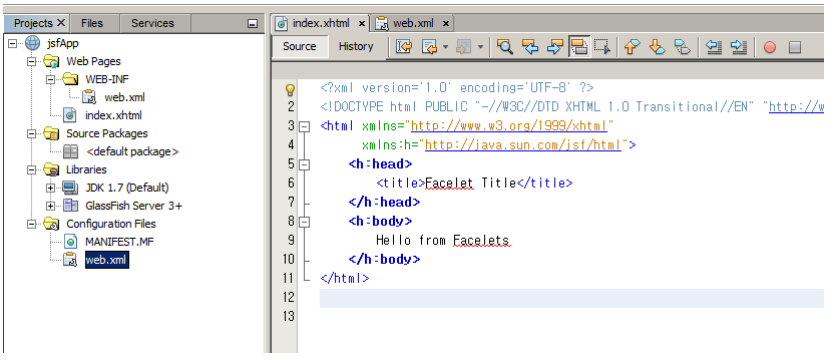
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
```

web.xml (cont'd)

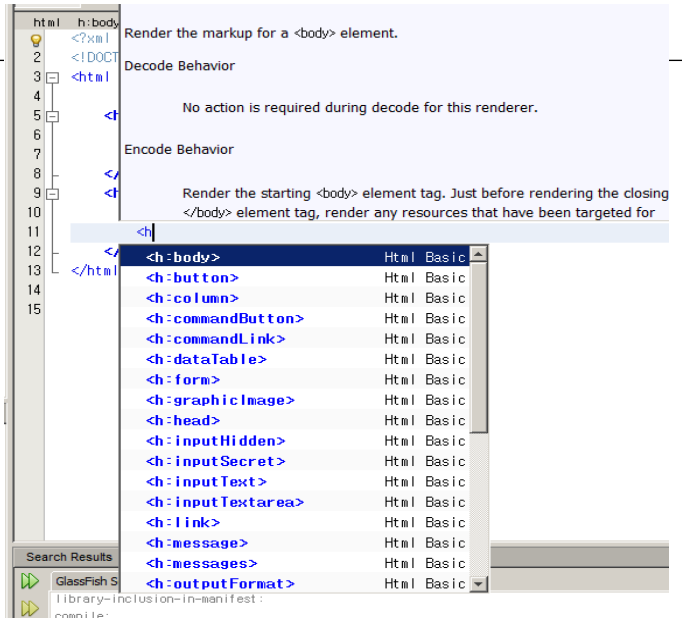
```
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>faces/index.xhtml</welcome-file>
  </welcome-file-list>
</web-app>
```

145

index.xhtml generated

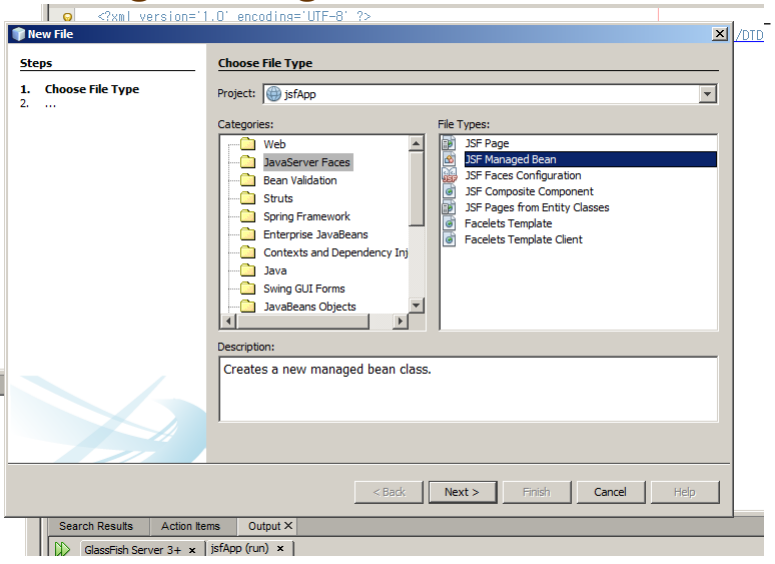


146



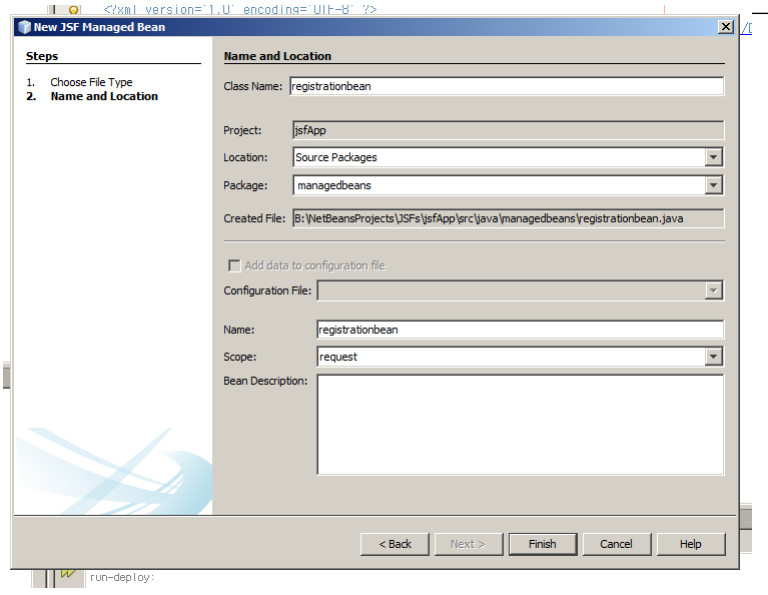
147

Creating managed Bean



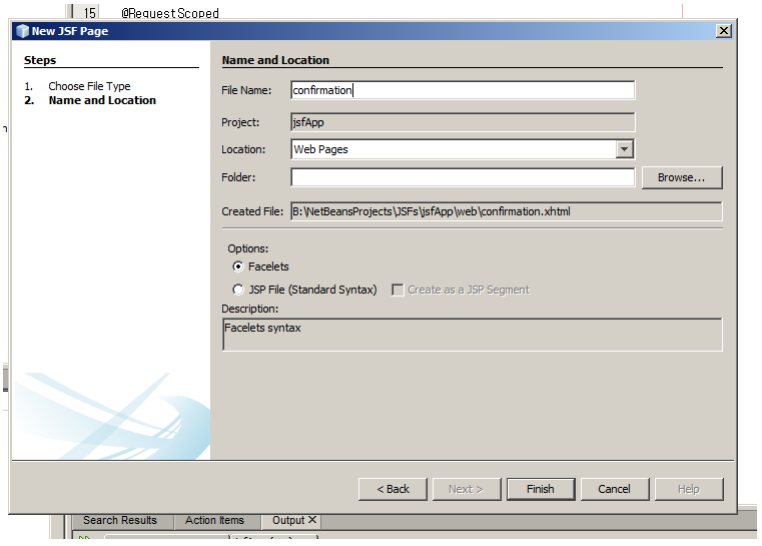
148

registrationbean



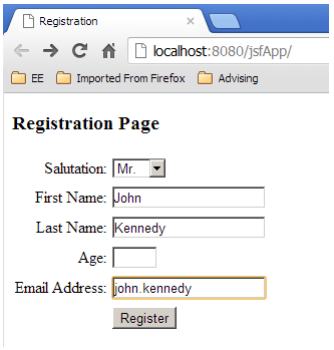
149

confirmation.xhtml

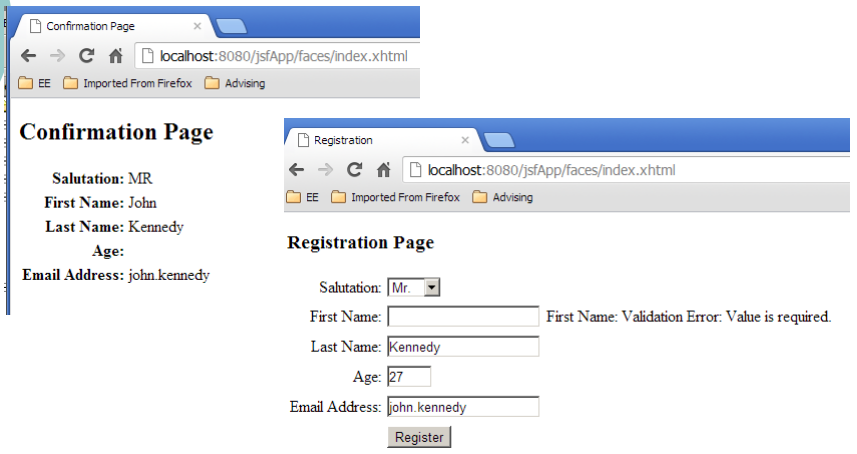


150

Run jsfApp



JSF Validation



Custom JSF Validator

Registration Page

Salutation: Mr.
First Name: John
Last Name: Kennedy
Age: 27
Email Address: johnkennedy@abc Email Address: not a valid email address
Register

localhost:8080/jsfApp/faces/index.xhtml
EE Imported From Firefox Advising

Confirmation Page

Salutation: MR
First Name: John
Last Name: Kennedy
Age: 27
Email Address: johnkennedy@abc.com

<h:message tag …/>

```
                value="#{registrationBean.lastName}" />
        <h:message for="lastName" />

        <h:outputLabel for="age" value="Age:"/>
        <h:inputText id="age" label="Age" size="2"
                value="#{registrationBean.age}"/>
        <h:message for="age"/>

        <h:outputLabel value="Email Address:" for="email"/>
        <h:inputText id="email" label="Email Address"
                required="true"
                value="#{registrationBean.email}">
          <f:validator validatorId="emailValidator"/>
        </h:inputText>
<!--            <h:message for="email" />-->

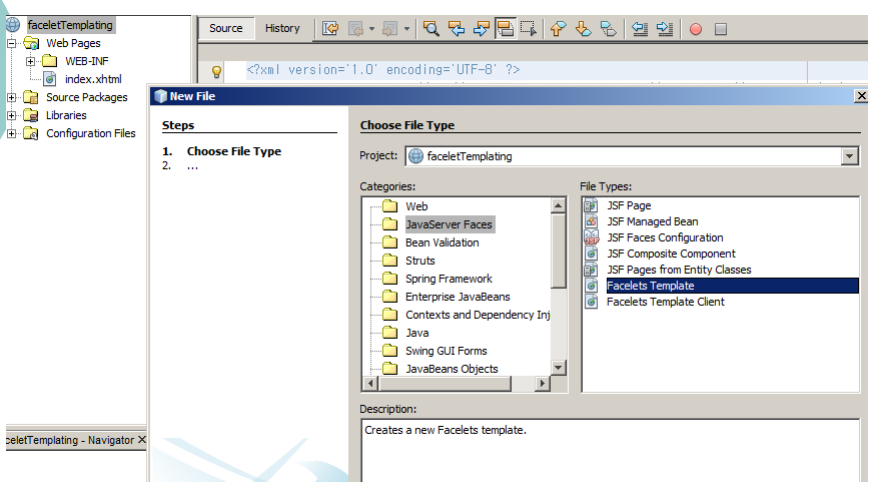
        <h:panelGroup/>
        <h:commandButton id="register" value="Register"
                        action="confirmation" />
```

Registration Page

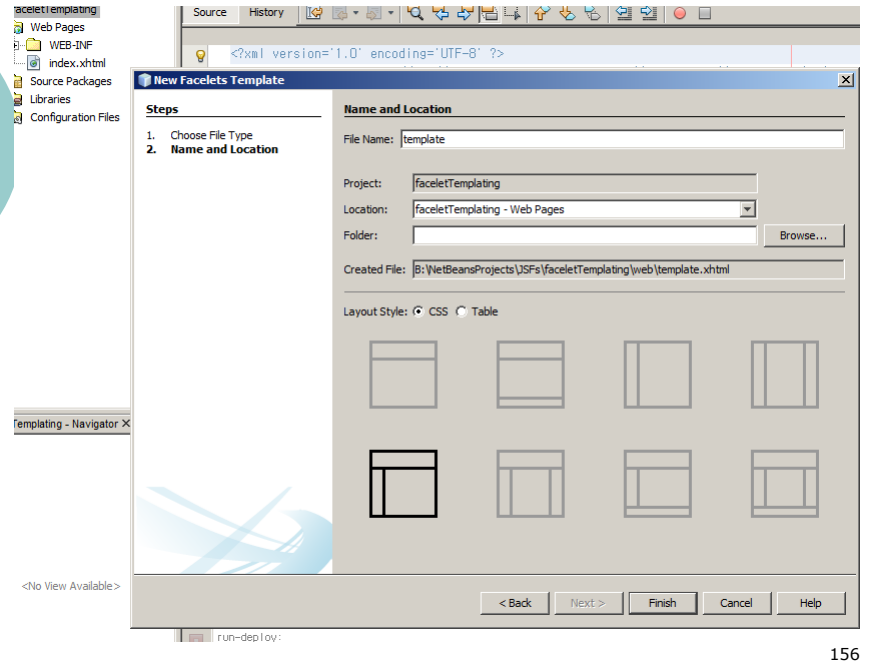
Salutation: Mr.
First Name: John
Last Name: Kennedy
Age: 27
Email Address: johnkennedy@abc
Register

- Email Address: not a valid email address

Facelets Templating

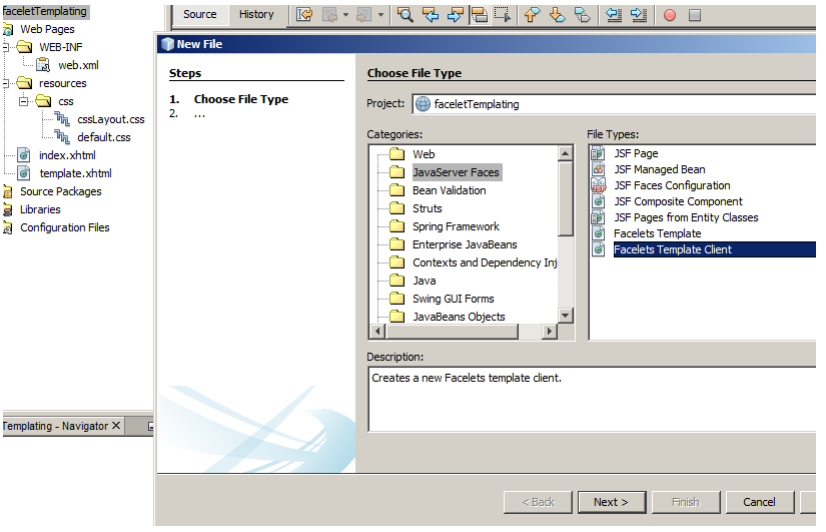


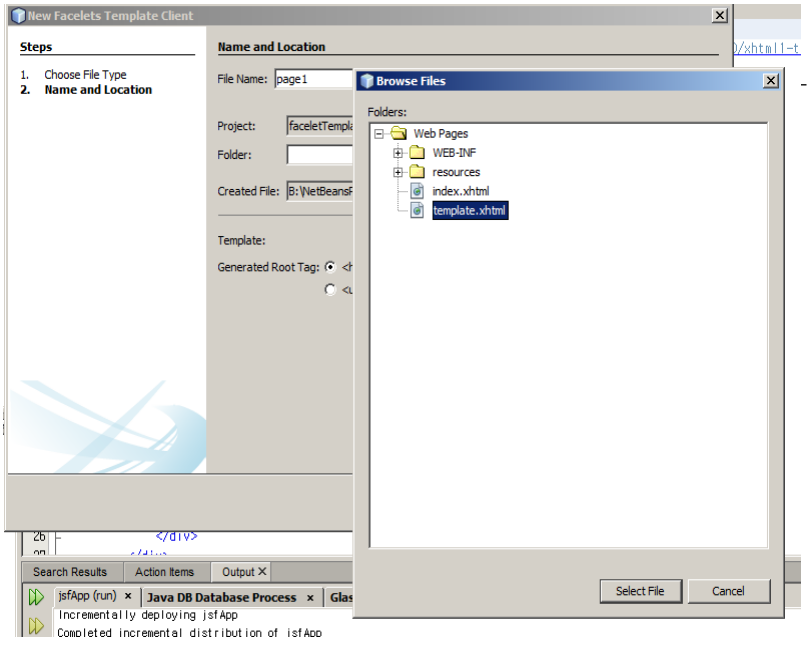
155



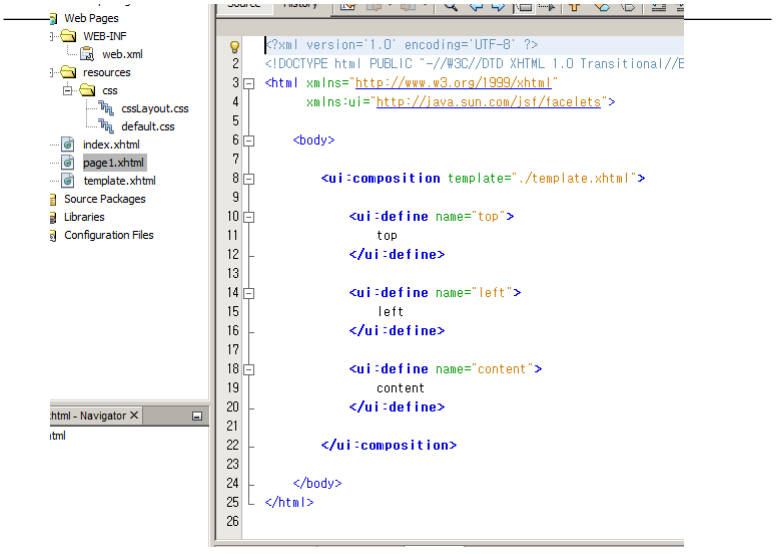
156

Using the template



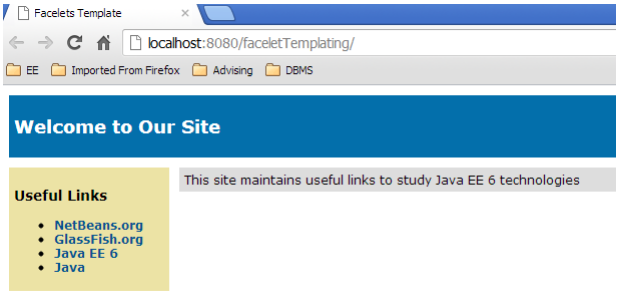


page1.xhtml (1)

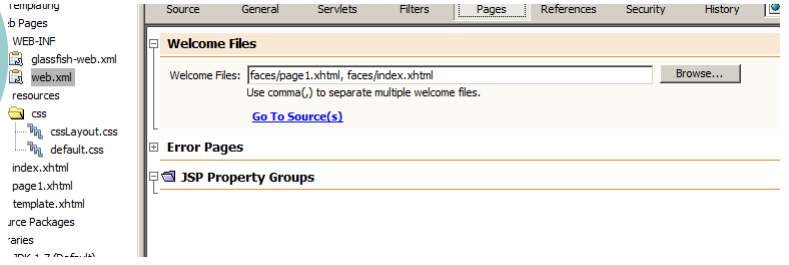


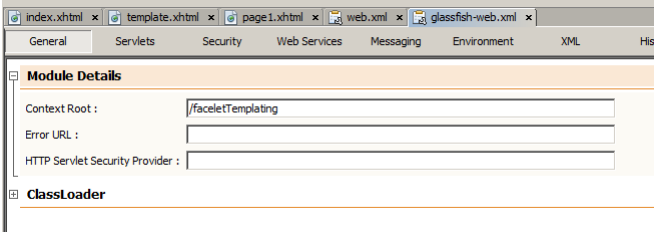
page1.xhtml (2)

Run faceletTemplating

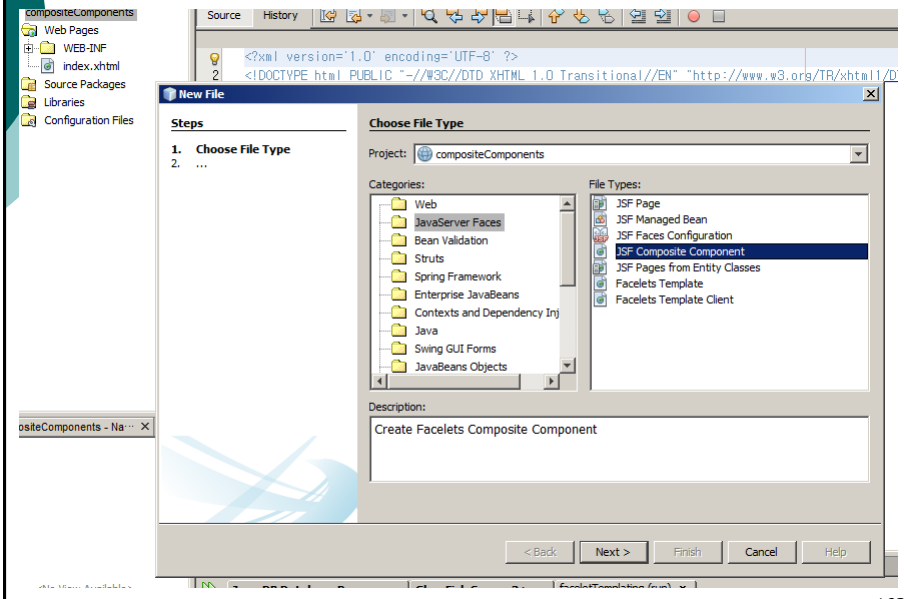


web.xml (faceletTemplating project)



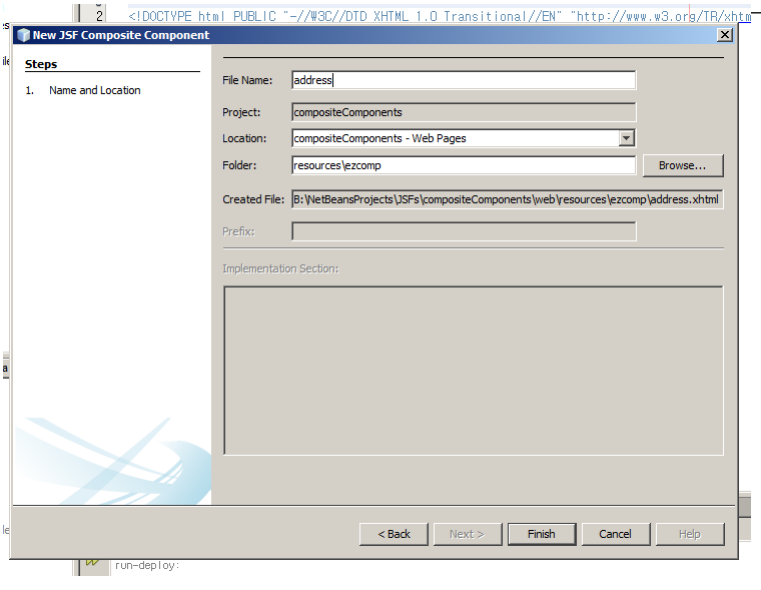


Composite Components



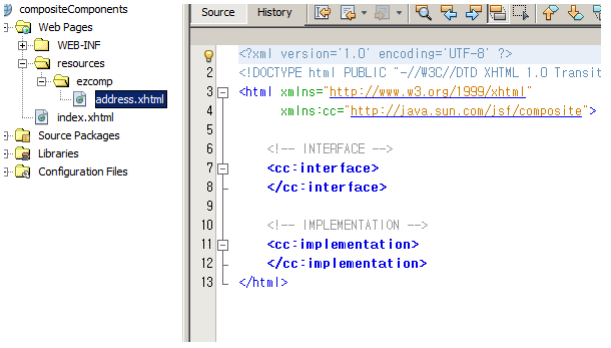
163

address Composite Component



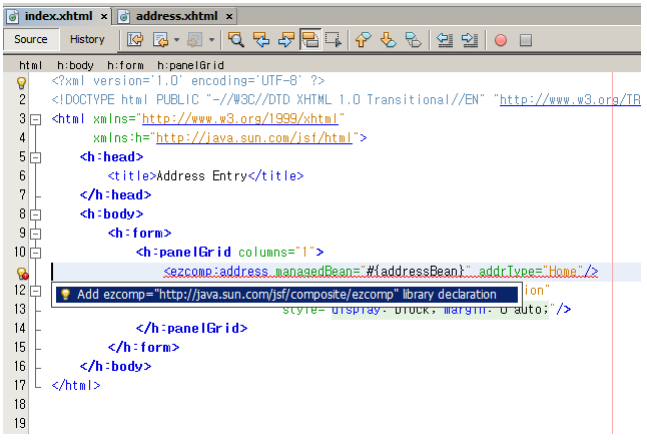
164

address.xhtml (1)

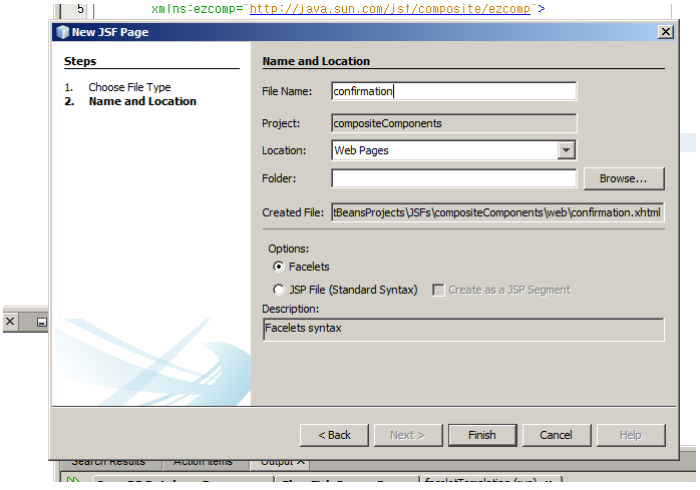


address.xhtml (2)

address.xhtml(3)



confirmation.xhtml

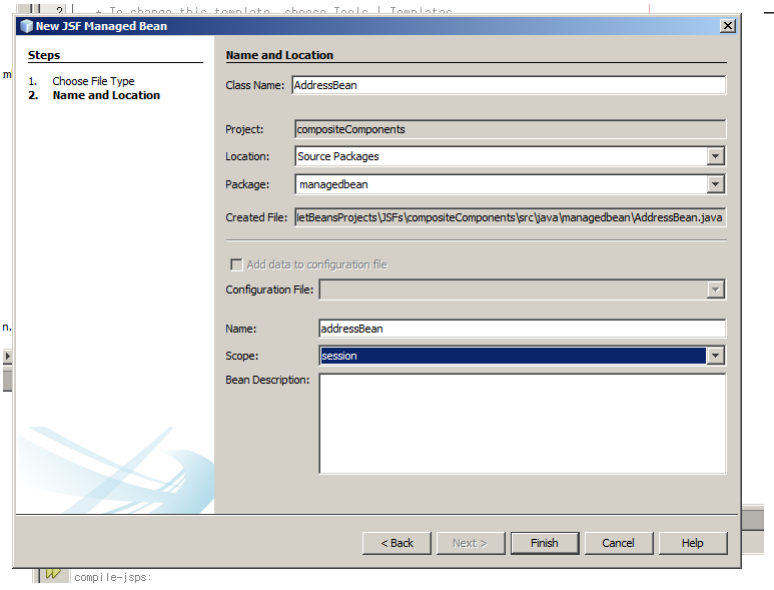


confirmation.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
   <h:head>
      <title>Confirmation Page</title>
   </h:head>
   <h:body>
      <h2>Address</h2>
      #{addressBean.line1}<br/>
      #{addressBean.line2}<br/>
      #{addressBean.city}<br/>
      #{addressBean.state}<br/>
      #{addressBean.zip}<br/>
   </h:body>
</html>
```

169

AddressBean.java



170

AddressBean.java (1)

```
@ManagedBean
@SessionScoped
public class AddressBean {

    /** Creates a new instance of AddressBean */
    public AddressBean() {
    }
    private String line1;
    private String line2;
    private String city;
    private String state;
    private String zip;

    public String getCity() {
        return city;
    }
```


AddressBean.java (2)

```
public void setCity(String city) {
        this.city = city;
    }

    public String getLine1() {
        return line1;
    }

    public void setLine1(String line1) {
        this.line1 = line1;
    }

    public String getLine2() {
        return line2;
    }

    public void setLine2(String line2) {
        this.line2 = line2;
    }
```

AddressBean.java (3)

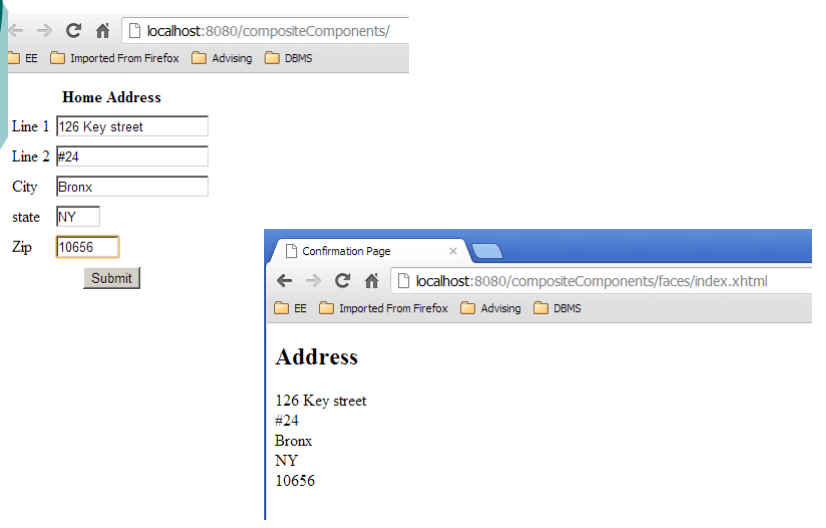
```
public String getState() {
    return state;
}

public void setState(String state) {
    this.state = state;
}

public String getZip() {
    return zip;
}

public void setZip(String zip) {
    this.zip = zip;
}
}
```

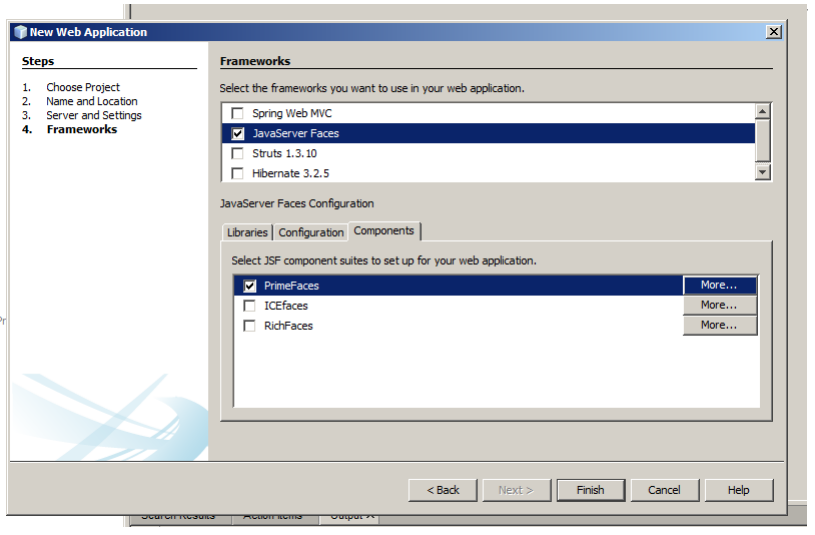

Run compositeComponents



JSF PrimeFaces

175

testPrimeFaces JSF project

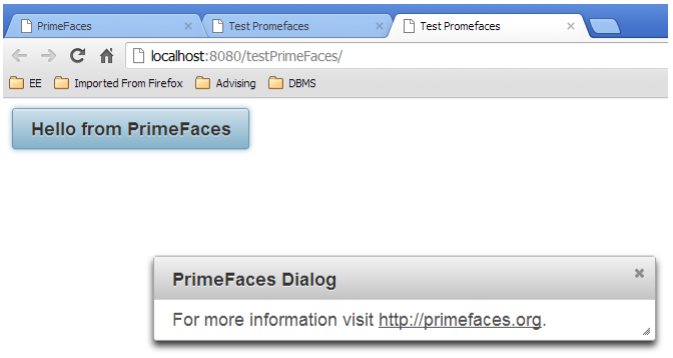


176

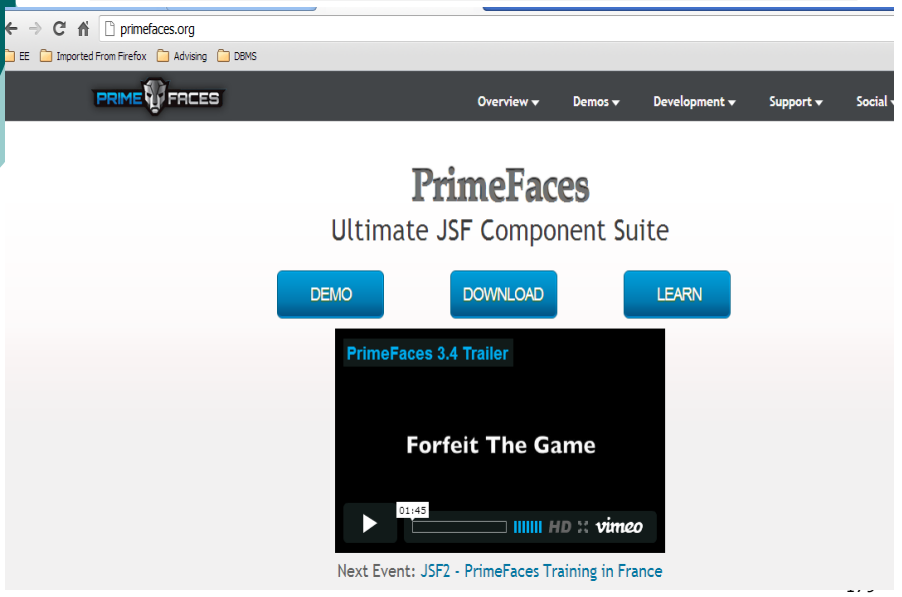
index.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:p="http://primefaces.org/ui"
    xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>Test PrimeFaces</title>
  </h:head>
  <h:body>
    <h:form><p:commandButton value="Hello from PrimeFaces" onclick="dlg1.show();" type="button" />
      <p:dialog header="PrimeFaces Dialog" widgetVar="dlg1" width="500">For more information visit
        <a href="http://primefaces.org">http://primefaces.org</a>.</p:dialog></h:form>
  </h:body>
</html>
```

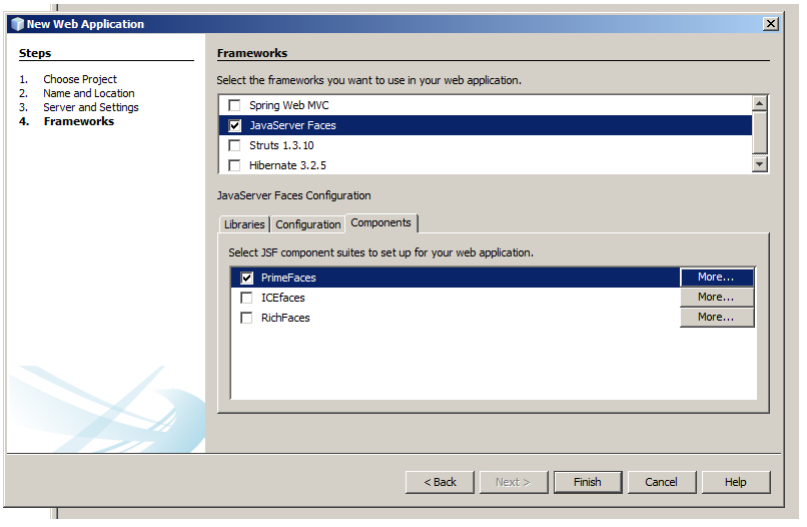

run testPrimeFaces



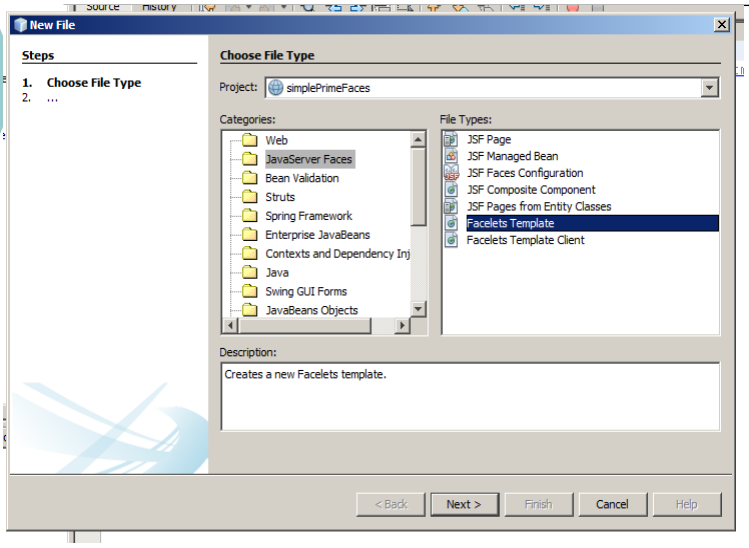
PrimeFaces.org



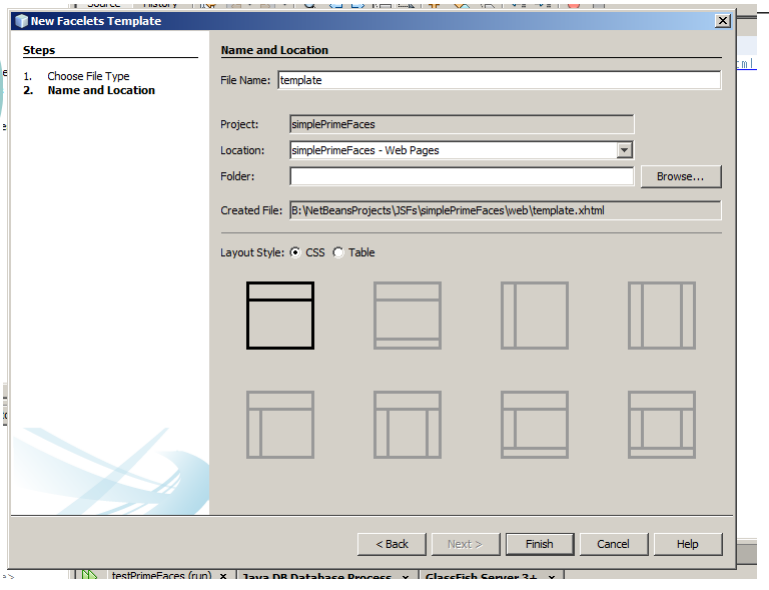
simplePrimeFaces project



Generate facelets template



template.xhtml (1)



template.xhtml (2)

```
<h:head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link href="./resources/css/default.css" rel="stylesheet" type="text/css" />
    <link href="./resources/css/cssLayout.css" rel="stylesheet" type="text/css" />
    <h:outputStylesheet library="css" name="custom.css"/>
    <title>Customer Information</title>
</h:head>

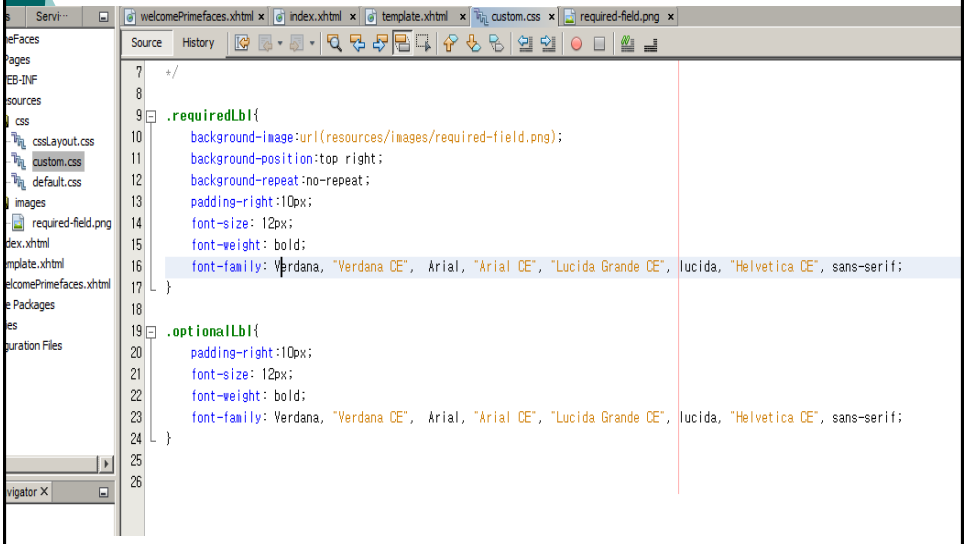
<h:body>

    <div id="top" class="top">
        <ui:insert name="top"><h2>Customer Information</h2></ui:insert>
    </div>

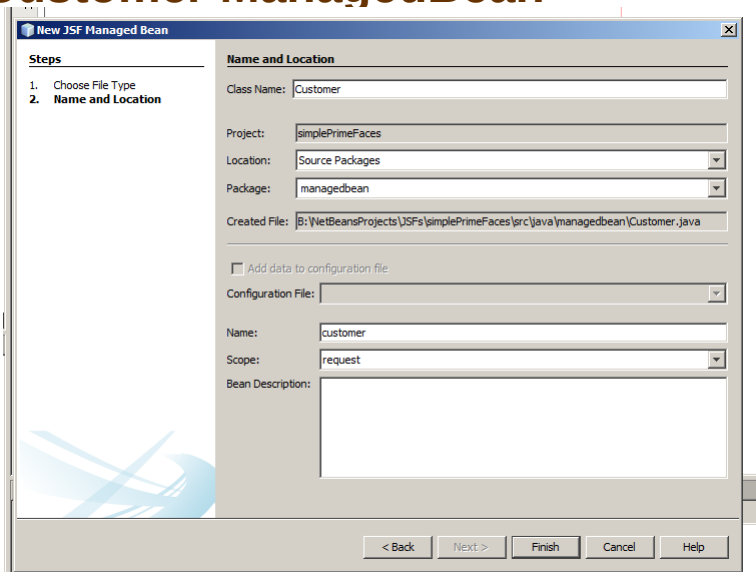
    <div id="content" class="center_content">
        <ui:insert name="content">Content</ui:insert>
    </div>

</h:body>
```


custom.css

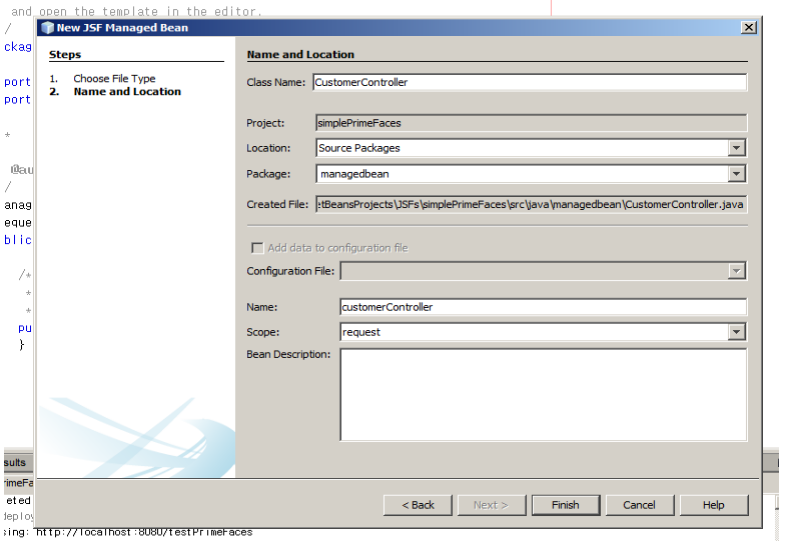


Customer ManagedBean



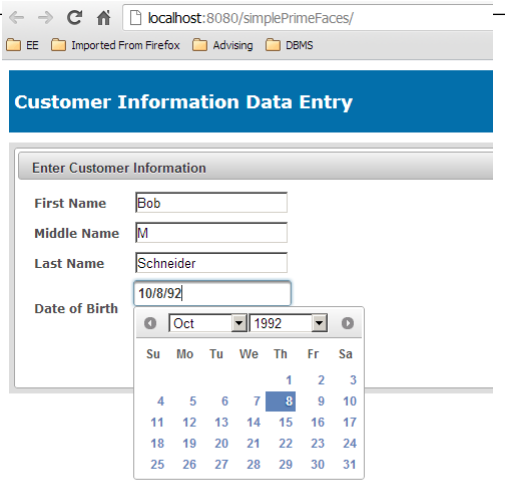
185

CustomerController.java

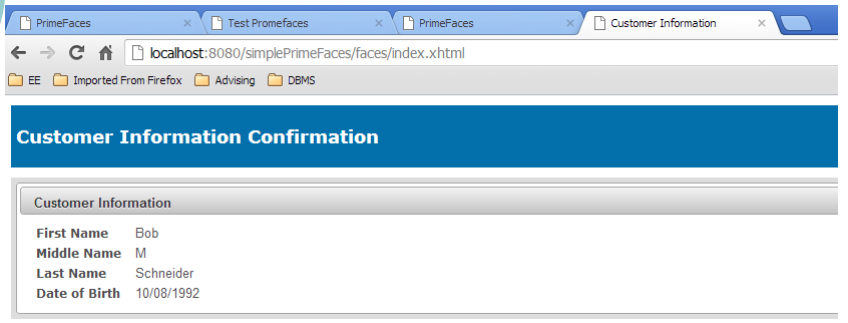


186

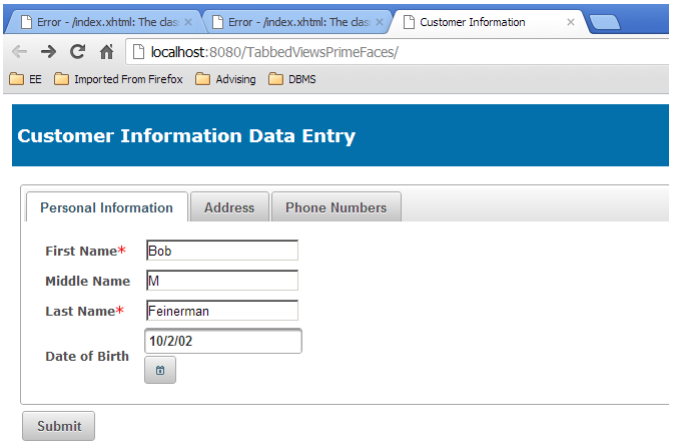
Run simplePrimeFaces (1)



Run simplePrimeFaces (2)

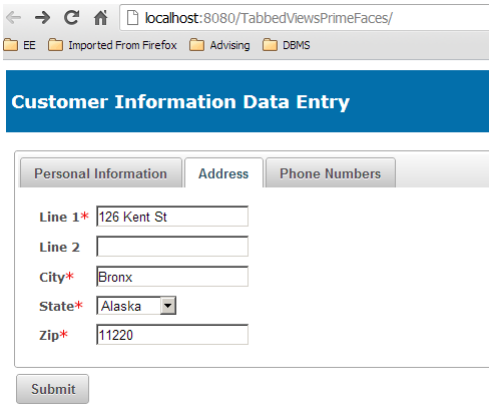


TabbedViewsPrimeFaces app (1)



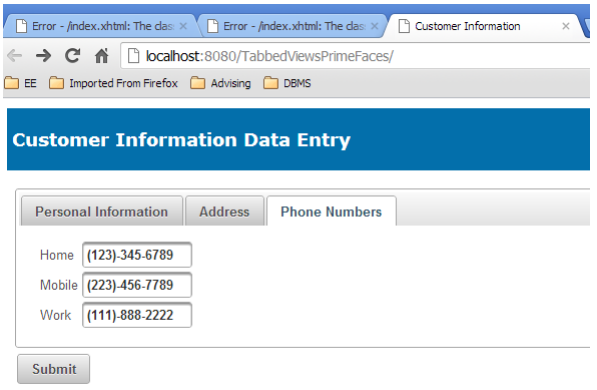
189

TabbedViewsPrimeFaces app (2)



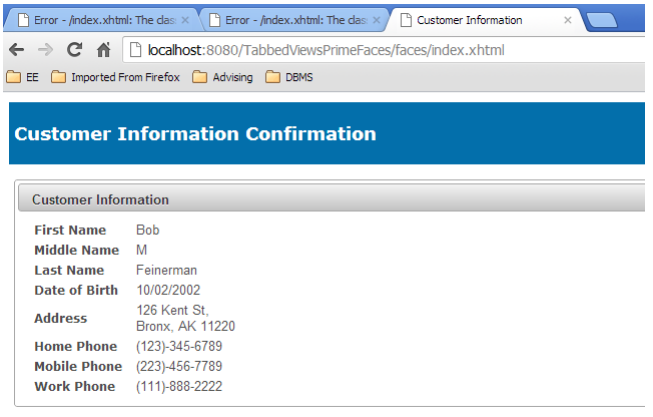
190

TabbedViewsPrimeFaces app (3)



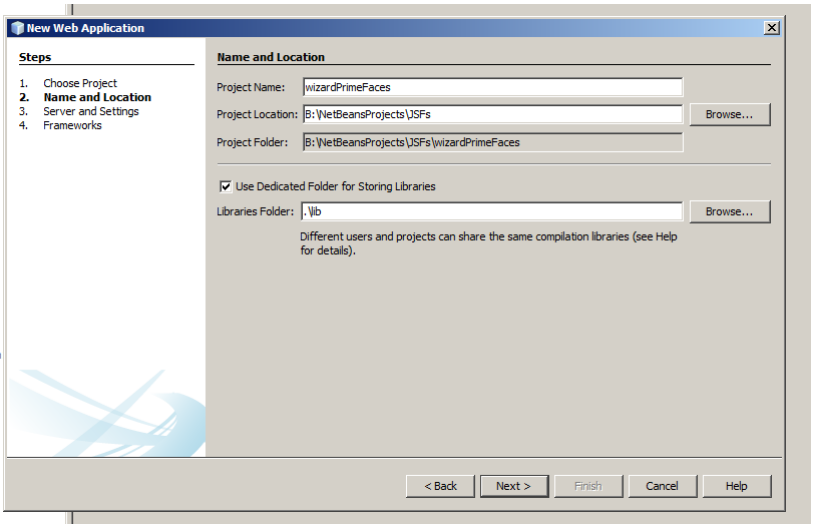
191

TabbedViewsPrimeFaces app (4)



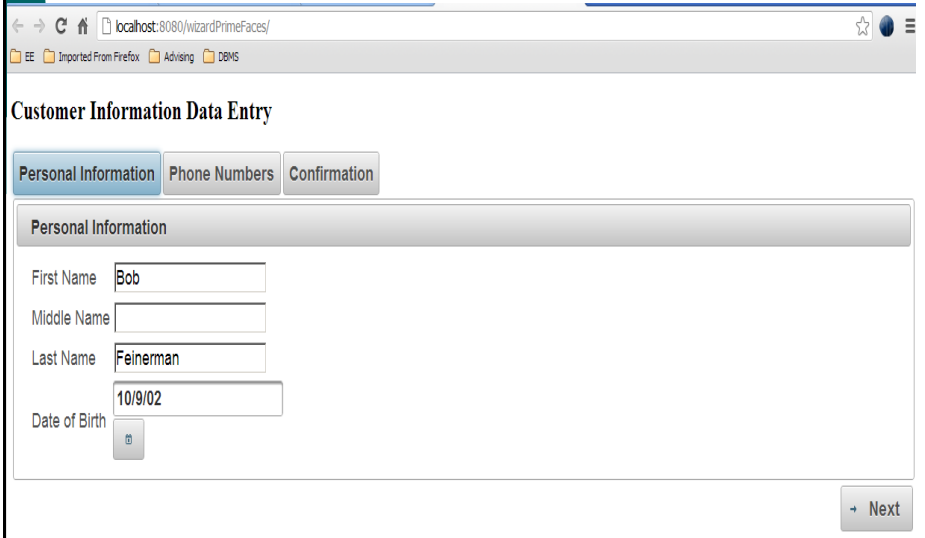
192

wizardPrimeFaces project



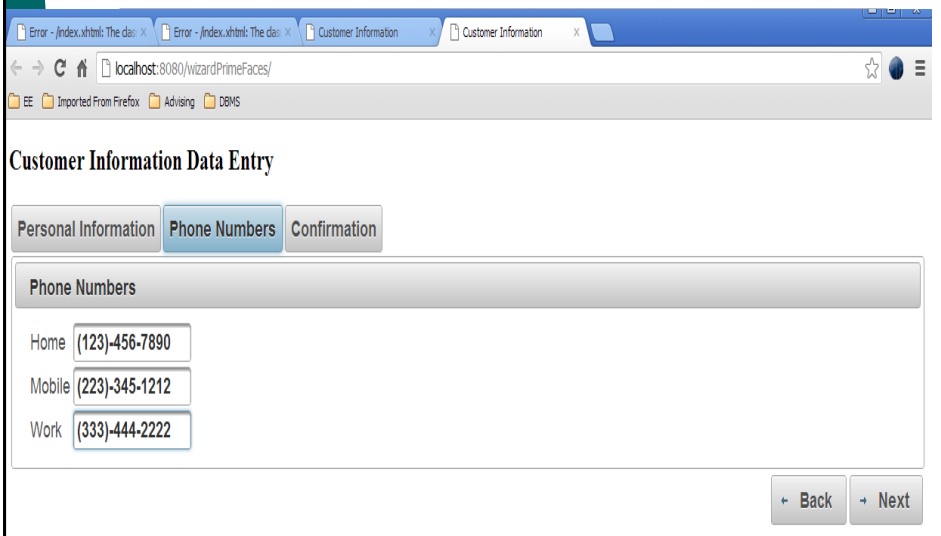
193

Run wizardPrimefaces app (1)



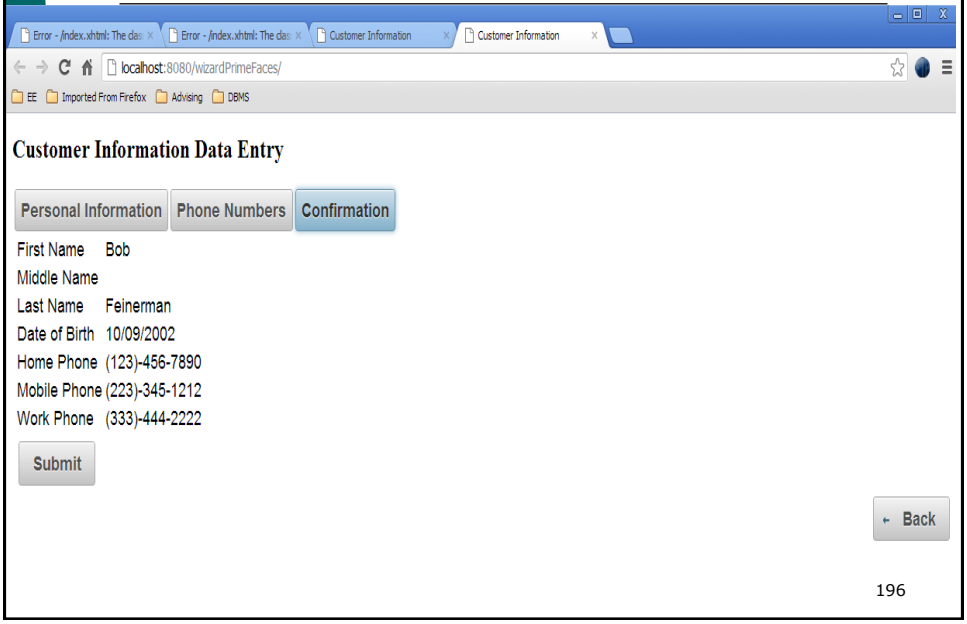
194

Run wizardPrimefaces app (2)



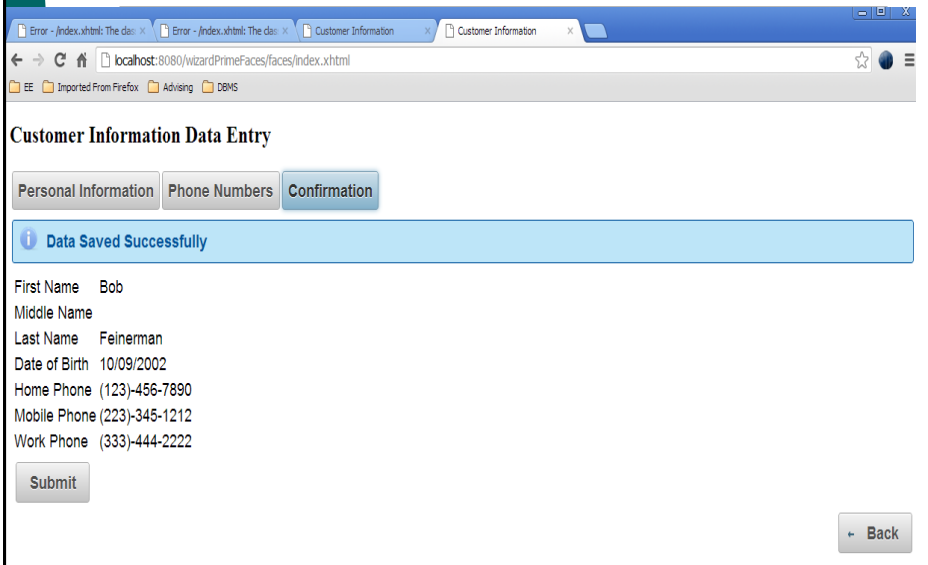
195

Run wizardPrimefaces app (3)



196

Run wizardPrimefaces app (4)



ConsultingAgency Project

Put it all together

- ConsultingAgency Application Development
  - Based on
    - JSF, ManagedBeans, Session EJBs, JPA, Entity classes based on
    - ConsultingAgency database schema
- Will be studied
  - in *JSF-Intro Part 2*