# JSF 2: Programming Basics

A Fast and Simplified
Overview of JSF 2 Development

**JSF 2.2 Version**

Originals of slides and source code for examples: http://www.coreservlets.com/JSF-Tutorial/jsf2/
Also see the PrimeFaces tutorial – http://www.coreservlets.com/JSF-Tutorial/primefaces/
and customized JSF2 and PrimeFaces training courses – http://courses.coreservlets.com/jsf-training.html

## Topics in This Section

- Simplified flow of control
- @ManagedBean and default bean names
- Default mappings for action controller return values
- Using bean properties to handle request parameters
- Common beginner problems

## Setup
(Review from Previous Section)

## Setup Summary

- **JAR files**
  - JSF 2.2 JAR file required
    - Omit it in Glassfish 4, JBoss 7, and other Java EE 7 servers
- **faces-config.xml**
  - For this entire section: empty body (start/end tags only)
    - This tutorial section uses Java-based annotations and default mappings of action controller values to results pages. Later tutorial sections will look at explicit values in faces-config.xml.
- **web.xml**
  - Must have a url-pattern for *.jsf (or other pattern you choose)
  - Usually sets PROJECT_STAGE to Development
- **Accessing file named some-page.xhtml**
  - Use URL some-page.jsf (matches url-pattern from web.xml)

## faces-config.xml

```
<?xml version="1.0"?>
<faces-config
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
     http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
    version="2.2">

</faces-config>
```

File is empty for now, but it has legal start and end tags so that later, when (if?) you start using faces-config, the file is already there and ready to use.

There will be no content inside the tags for any of the examples in this section. All examples in this section use default bean names (derived from the bean's class name with the first letter changed to lower case) and default results pages (derived from the action controller's return values).

Do not type face-config.xml or web.xml by hand. Instead, copy from the jsf-blank project included with this tutorial.

## web.xml (Slightly Simplified)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app …  version="3.0">
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
  </servlet-mapping>
  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
  <welcome-file-list>
    <welcome-file>index.jsf</welcome-file>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

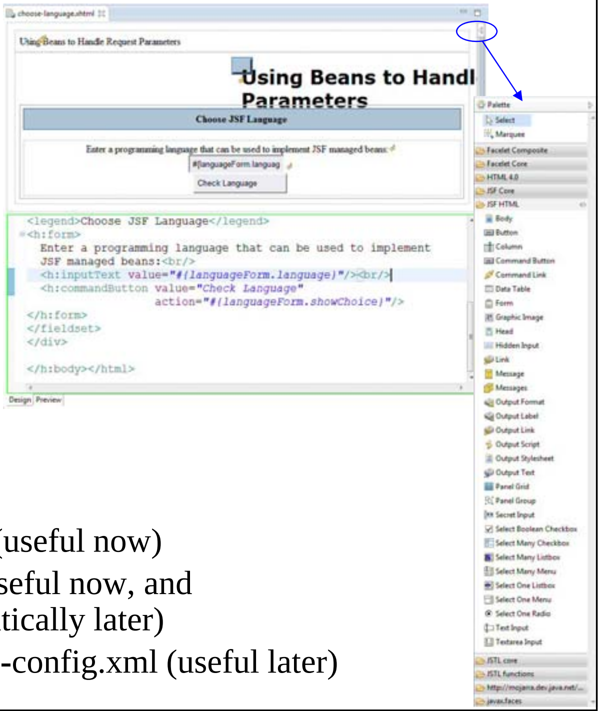
The real file is blah.xhtml, but the URL is blah.jsf. You can change this to *.faces or *.fubar, in which case the URL for blah.xhtml would be blah.faces or blah.fubar.

Means that you get extra debugging support. In particular, a Java action that has no corresponding file results in an error message, rather than the original page being displayed with no hint about the problem.

Means that you can put index.xhtml in the home directory of your app, use the URL http://*hostname*/*appname*/, and it will act as though you did http://*hostname*/*appname*/index.jsf.

## Eclipse Support for JSF 2.2

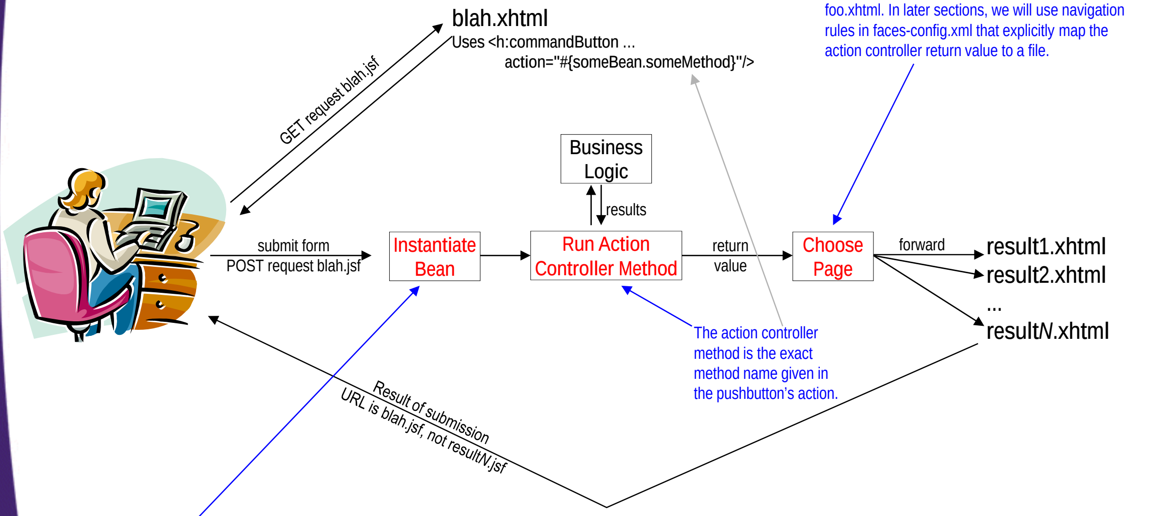
- **Add JSF 2 project facet**
  - R-click project, Properties, Project Facets, check "JavaServer Faces 2.2"
    - The downloadable Eclipse projects from JSF 2 tutorial at coreservlets.com *already* have this facet set.
  - The first time you do it, you will have to give location of the JSF 2.2 JAR file
    - Coreservlets sample projects use JSF 2.2 JAR file already
- **Benefits**
  - Visual previews of .xhtml files (useful now)
  - Palette of drag-and-drop tags (useful now, and will pick up PrimeFaces automatically later)
  - Lots of support for editing faces-config.xml (useful later)



```
<legend>Choose JSF Language</legend>
<h:form>
  Enter a programming language that can be used to implement
  JSF managed beans:<br/>
  <h:inputText value="#{languageForm.language}"/><br/>
  <h:commandButton value="Check Language"
                   action="#{languageForm.showChoice}"/>
</h:form>
</fieldset>
</div>

</h:body></html>
```

# Basic Structure of JSF 2 Apps

## JSF Flow of Control (Highly 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. In later sections, we will use navigation rules in faces-config.xml that explicitly map the action controller return value to a file.

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

The pushbutton said action="#{someBean.someMethod}". So, instantiate bean whose name is someBean. In this section, we will declare bean with @ManagedBean, so that means to instantiate bean whose class name is SomeBean. In later sections, we will see that the bean could be session-scoped (or have other scopes), so this will be called "Find Bean" instead of "Instantiate Bean".

## Basic Structure of Facelets Pages

```
<!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://xmlns.jcp.org/jsf/html">
<h:head>
…
</h:head>
<h:body>
…
<h:form>
…
</h:form>
…
</h:body>
</html>
```

JSF 1 programmers: You use "facelets" – pages that use xhtml syntax – for all JSF 2 pages; you never use old-style JSP syntax. You always have xmlns:h…, h:head, h:body, and (for input forms) h:form. In later sections we will see that you sometimes also have xmlns:f... and/or xmlns:ui… Results pages that do not also contain input elements can omit the h:form part. No @taglib entries needed.

JSF 2.0 and 2.1 programmers: note that the host of java.sun.com has been replaced by xmlns.jcp.org, but the old name still works for backward compatibility.

All: remember that the URL does not match the real filename: you use blah.xhtml for the files, but blah.jsf for the URLs (or whatever ending matches the url-pattern in web.xml).

Finally, note that the "samples" folder of the jsf-blank project has a simple template file that contains the code shown here. Use that as a starting point for your own .xhtml files, rather than typing this all in by hand.

## Basic Structure of Managed Beans

```
@ManagedBean
public class SomeBean {
  private String someProperty;

  public String getSomeProperty() { … }

  public void setSomeProperty() { … }

  public String actionControllerMethod() { … }

  // Other methods
}
```

Managed beans are Java classes that are declared with @ManagedBean or listed in faces-config.xml. More details will be given in the next tutorial sections, but for now the main points are:

- They are usually POJOs (they implement no special interfaces, and most methods have no JSF-specific argument or return types).
- They have pairs of getter and setter methods corresponding to each input element in the form.
- They have an action controller method 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.
- They also typically have placeholders for derived properties – information that will be computed based on the input data. This part is omitted for now, but more on this in the upcoming lecture on managed beans.

# @ManagedBean Basics

## 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 page that had 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="#{navigator.choosePage}"/>
  - A bean
    - Class: Navigator (bean name above except for case)
    - @ManagedBean annotation
    - choosePage method returns 3 possible Strings
      - "page1", "page2", or "page3"
  - Three results pages
    - Names match return values of choosePage method
      - page1.xhtml, page2.xhtml, and page3.xhtml

## start-page.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://xmlns.jcp.org/jsf/html">
<h:head>…</h:head>
<h:body>
…
<fieldset>
<legend>Random Results Page</legend>
<h:form>
  Press button to get one of three possible results pages.
  <br/>
  <h:commandButton value="Go to Random Page"
                   action="#{navigator.choosePage}"/>
</h:form>
</fieldset>
…
</h:body></html>
```

This means that when you press button, JSF instantiates bean whose name is navigator and then runs the choosePage method. This is same format as in JSF 1.*x*, but here name of bean is automatically derived from Java class name.

## Navigator.java

```
package coreservlets;

import javax.faces.bean.*;

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

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

Declares this as managed bean, without requiring entry in faces-config.xml as in JSF 1.*x*.

Since no name given, name is class name with first letter changed to lower case (i.e., navigator). You can also do @ManagedBean(name="someName"). See later section.

Since no scope given, it is request scoped. You can also use an annotation like @SessionScoped. See later section.

The randomElement method just uses Math.random to return an element from the array at random. Source code is in the downloadable Eclipse project.

Since there are no explicit navigation rules in faces-config.xml, these return values correspond to page1.xhtml, page2.xhtml, and page3.xhtml (in same folder as page that has the form).

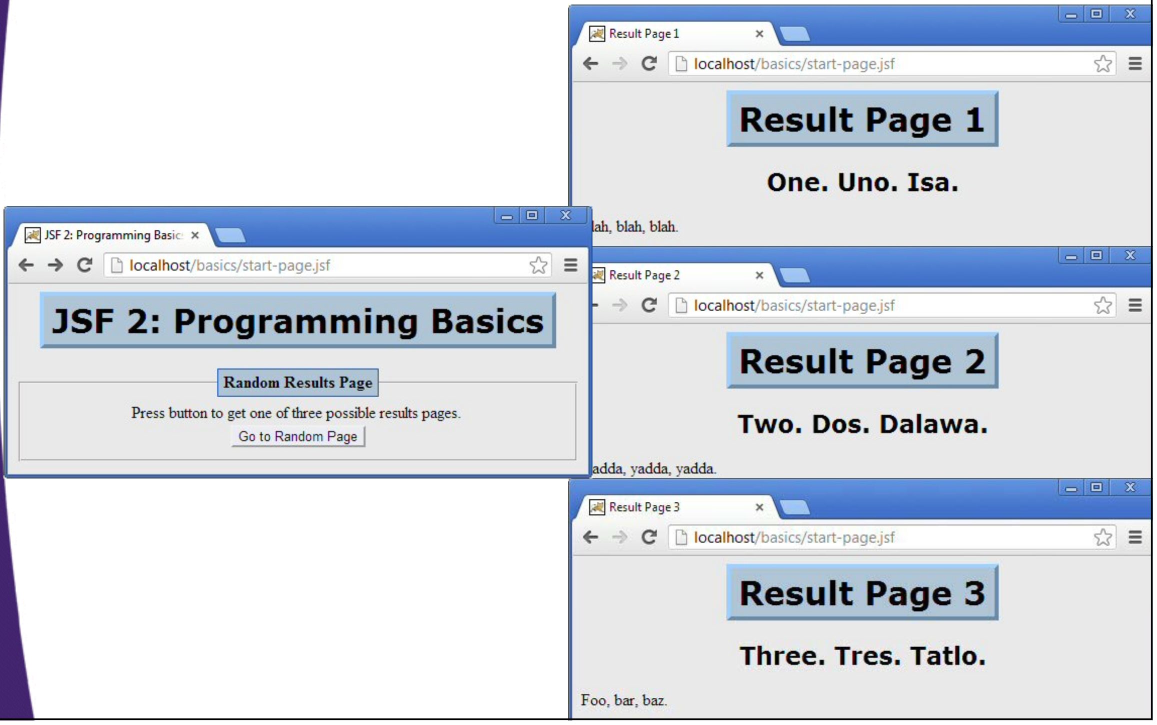
## page1.xhtml

```
<!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://xmlns.jcp.org/jsf/html">
<h:head><title>Result Page 1</title>
<link href="./css/styles.css"
      rel="stylesheet" type="text/css"/>
</h:head>
<h:body>

<table class="title">
  <tr><th>Result Page 1</th></tr>
</table>
<p/>
<h2>One. Uno. Isa.</h2>
<p>Blah, blah, blah.</p>

</h:body></html>
```

I don't actually have any dynamic code in this simplistic example, but it is a good idea to plan ahead and always include h:head and h:body.

page2.xhtml and page3.xhtml are similar.

## Results



JSF 2: Programming Basics

localhost/basics/start-page.jsf

# JSF 2: Programming Basics

Random Results Page

Press button to get one of three possible results pages.

Go to Random Page

Result Page 1

localhost/basics/start-page.jsf

# Result Page 1

## One. Uno. Isa.

lah, blah, blah.

Result Page 2

localhost/basics/start-page.jsf

# Result Page 2

## Two. Dos. Dalawa.

adda, yadda, yadda.

Result Page 3

localhost/basics/start-page.jsf

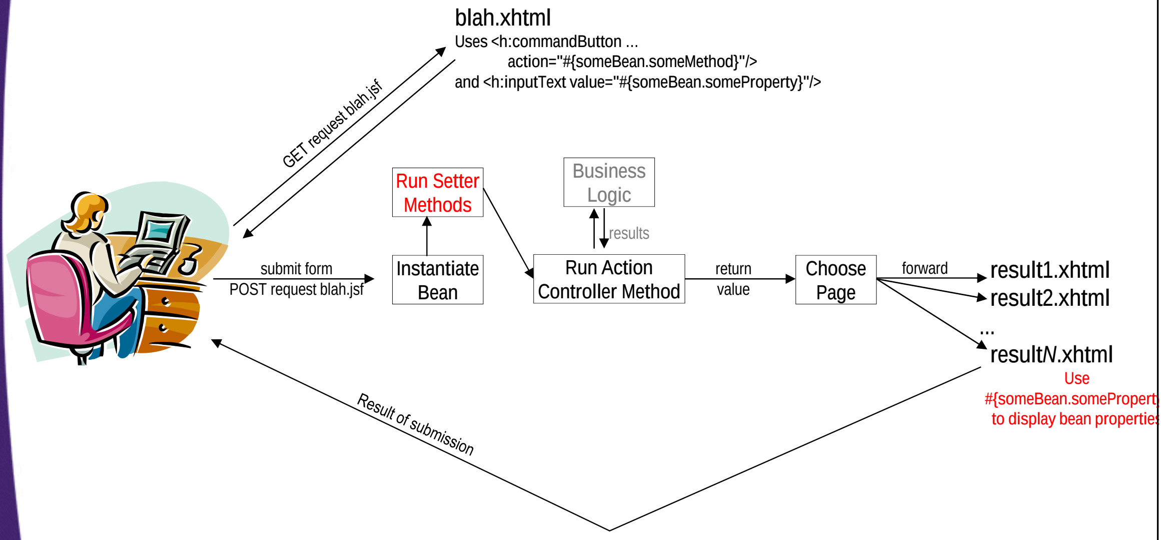
# Result Page 3

## Three. Tres. Tatlo.

Foo, bar, baz.

# Using Beans to Handle Request Parameters

## JSF Flow of Control (Updated but Still Simplified)



## Main Points

- **Input values correspond to bean properties**
  – <h:inputText value="#{someBean.someProp}"/>
    - When form is submitted, takes value in textfield and passes it to setSomeProp.
      – Validation and type conversion (if any) is first. See later section.
    - When form is displayed, calls getSomeProp(). If value is other than null or empty String, puts value in field. See later section.
  – Same behavior as with bean properties in JSF 1.*x*
- **Beans are request scoped by default**
  – Bean is instantiated twice: once when form is initially displayed, then again when form is submitted.
  – Same behavior as with request-scoped beans in JSF 1.*x*.
- **Can use #{bean.someProp} directly in output**
  – Means to output result of getSomeProp()
    - Instead of <h:outputText value="#{bean.someProp}"/> as in JSF 1

## Example

- **Idea**
  - Enter name of a programming language
  - Get one of
    - Error page: no language entered
    - Warning page: language cannot be used for JSF
      - Needs to output the language the user entered
    - Confirmation page: language is supported by JSF
- **New features you need**
  - Bean
    - Properties corresponding to request parameters
  - Input form
    - <h:inputText value="#{languageForm.language}"/>
  - Results pages
    - #{languageForm.language}   (for warning page)

## choose-language.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://xmlns.jcp.org/jsf/html">
…
<h:body>
…
<fieldset>
<legend>Choose JSF Language</legend>
<h:form>
  Enter a programming language that can be used to implement
  JSF managed beans:<br/>
  <h:inputText value="#{languageForm.language}"/><br/>
  <h:commandButton value="Check Language"
                   action="#{languageForm.showChoice}"/>
</h:form>
</fieldset>
…
</h:body></html>
```

When form is submitted, languageForm is instantiated and textfield value is passed to setLanguage.

Then, the showChoice method is called to determine the results page.

The value of h:inputText actually plays a dual role. When form is first displayed, languageForm is instantiated and getLanguage is called. If the value is non-empty, that result is the initial value of the textfield. Otherwise, the textfield is initially empty. When the form is submitted, languageForm is reinstantiated (assuming request scope) and the value in the textfield is passed to setLanguage. More on this dual behavior in the next tutorial section, but for now just be aware that your bean must have both getLanguage and setLanguage methods.

## LanguageForm.java (Top)

```
package coreservlets;

import javax.faces.bean.*;

@ManagedBean
public class LanguageForm {
  private String language;

  public String getLanguage() {
    return(language);
  }

  public void setLanguage(String language) {
    this.language = language.trim();
  }
```

This will be automatically called by JSF when form is submitted.

Using #{languageForm.language} in the results page corresponds to the getLanguage method. Using <h:inputText value="#{languageForm.language}"/> in the input form means the textfield value will be passed to the setLanguage method. The names of instance variables (if any) is irrelevant. The next lecture will give the full rules for mapping the short form to the method names, but the simplest and most common rule is to drop " get" or "set" from the method name, then change the next letter to lower case.

## LanguageForm.java (Continued)

```
  public String showChoice() {
    if (isMissing(language)) {
      return("missing-language");
    } else if (language.equalsIgnoreCase("Java") ||
               language.equalsIgnoreCase("Groovy")) {
      return("good-language");
    } else {
      return("bad-language");
    }
  }

  private boolean isMissing(String value) {
    return((value == null) || (value.trim().isEmpty()));
  }
}
```

The action of h:commandButton is this exact method name, rather than a shortcut for a pair of getter and setter methods as with h:inputText.

## missing-language.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://xmlns.jcp.org/jsf/html">
<h:head>
…
</h:head>
<h:body>

<table class="title">
  <tr><th>Missing Language</th></tr>
</table>
<h2>Duh! You didn't enter a language!
(<a href="choose-language.jsf">Try again</a>)</h2>
<p>Note that using separate error pages for missing
input values does not scale well to real applications.
The later section on validation shows better approaches.</p>

</h:body></html>
```

## bad-language.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://xmlns.jcp.org/jsf/html">
<h:head>
…
</h:head>
<h:body>

<table class="title">
  <tr><th>Bad Language</th></tr>
</table>
<h2>Use #{languageForm.language} in JSF?
Be serious!</h2>

</h:body></html>
```

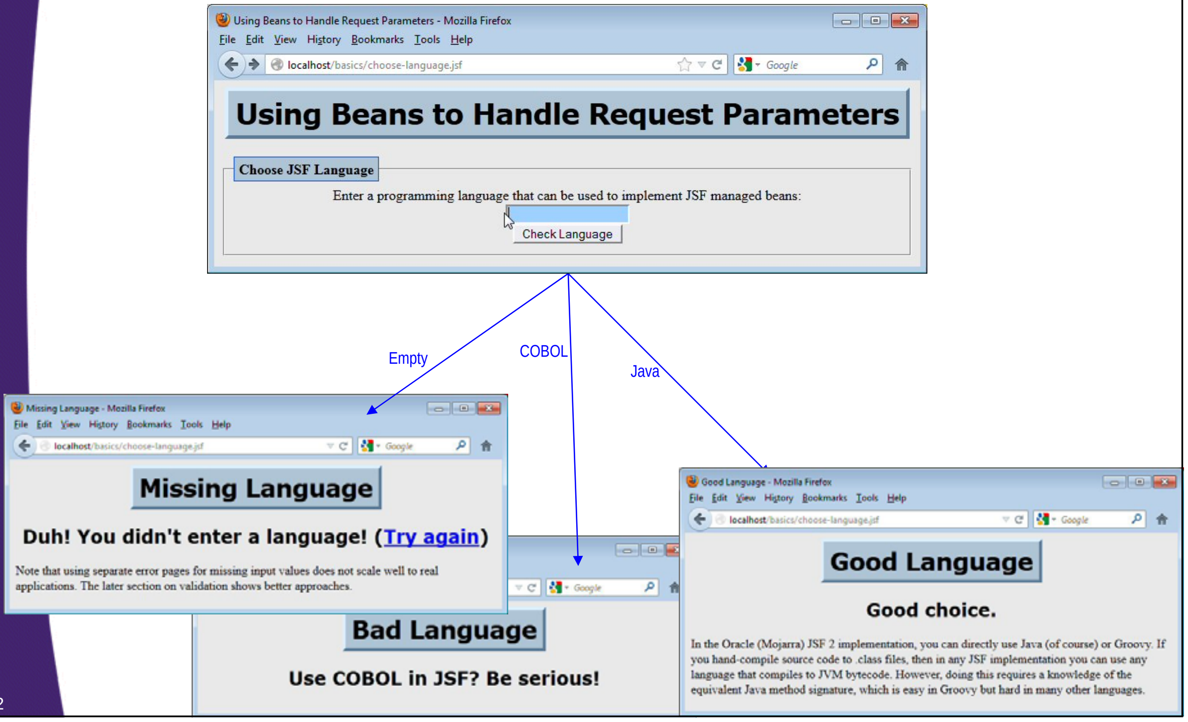
Calls getLanguage and outputs the result.

In JSF 2.*x* you can use #{result} instead of <h:outputText value="#{result}"/> as was needed in JSF 1.*x*. Both approaches escape HTML characters, so you don't have to worry about the user entering HTML tags. Therefore, use the shorter approach shown here unless you need one of the options to h:outputText like escape (with a value of false), rendered (with a computed value), id, converter, etc. These are covered in later lectures.

## good-language.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://xmlns.jcp.org/jsf/html">
<h:head>
…
</h:head>
<h:body>

<table class="title">
  <tr><th>Good Language</th></tr>
</table>
<h2>Good choice.</h2>
<p>In the Oracle (Mojarra) JSF 2 implementation, … </p>

</h:body></html>
```

## Results


Using Beans to Handle Request Parameters - Mozilla Firefox
File Edit View History Bookmarks Tools Help
localhost/basics/choose-language.jsf
Check Language
Empty
COBOL
Java
Missing Language - Mozilla Firefox
Good Language - Mozilla Firefox

Using Beans to Handle Request Parameters

Choose JSF Language

Enter a programming language that can be used to implement JSF managed beans:

Missing Language

Duh! You didn't enter a language! (Try again)

Note that using separate error pages for missing input values does not scale well to real applications. The later section on validation shows better approaches.

Bad Language

Use COBOL in JSF? Be serious!

Good Language

Good choice.

In the Oracle (Mojarra) JSF 2 implementation, you can directly use Java (of course) or Groovy. If you hand-compile source code to .class files, then in any JSF implementation you can use any language that compiles to JVM bytecode. However, doing this requires a knowledge of the equivalent Java method signature, which is easy in Groovy but hard in many other languages.

# Interactive Example

## Simplified App from Scratch (in 5-10 Minutes)

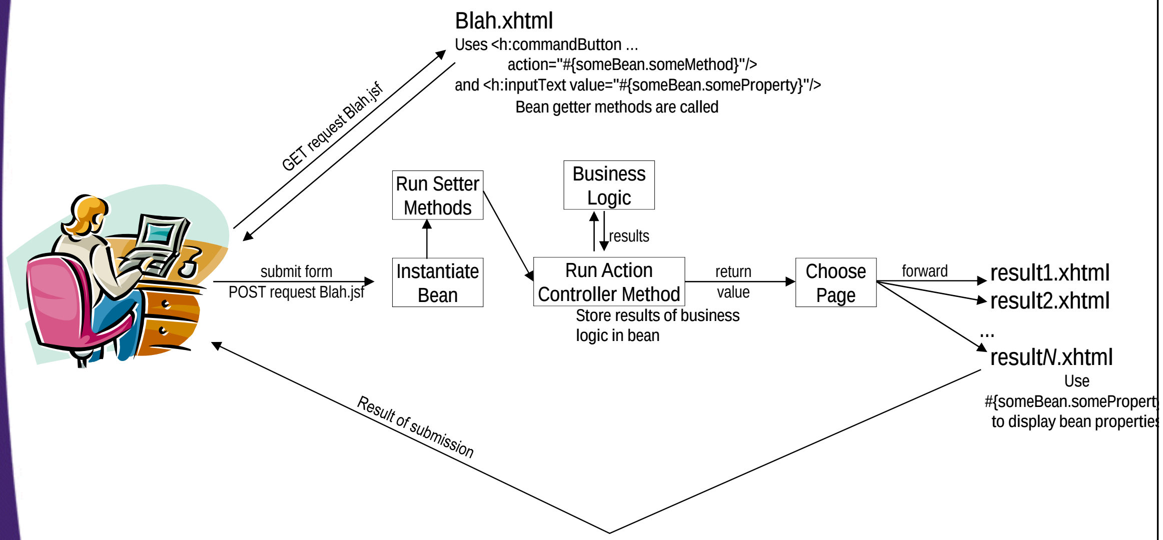
- **Make new project based on jsf-blank**
  - Be sure you can see blank index.jsf page
- **Insert form that has button**
  - Pressing button results in error that bean not found
- **Make Java class with action controller to respond to button**
  - Pressing button results in error that page is not found
- **Make results pages**
- **Add textfield to form**
- **Extend Java class to have get/set methods**
- **Output the form values in results pages**

# Wrap-Up

## Common Beginner Problems

- **JSF tags appear to be ignored**
  - You entered URL ending in *blah*.xhtml instead of *blah*.jsf
  - Or, your filename was index.html instead of index.xhtml
- **Error message about null source**
  - You have XML syntax error in main page. For example:
    - <h:commandButton action="…"> (instead of <h:commandButton action="…"/> )
    - Note that Eclipse is quite helpful in finding XML syntax errors
- **Error message that view cannot be restored**
  - You went a very long time (e.g., during the lecture) without reloading the page or pressing a button.
  - Solution: copy the URL, restart browser, and paste in URL
- **New Java class not found by JSF**
  - If you add a *new* class that uses @ManagedBean, you must restart the server. (Also true if you edit web.xml or faces-config.xml, but we aren't doing either of those yet.)

## Highly Simplified JSF Flow of Control



## Summary

- **Input pages with forms ("facelets" pages)**
  - Declare h: namespace, use h:head, h:body, h:form
    - Use template from "samples" folder of jsf-blank
- **Java code: managed beans**
  - Declare with @ManagedBean
    - Bean name is class name with first letter in lower case
  - Getter and setter for each input element
      - Form: <h:inputText value="#{beanName.propertyName}"/>
  - Action controller method
      - Form: <h:commandButton action="#{beanName.methodName}"/>
    - Return values become base names of results pages
- **Results pages**
  - Declare h: namespace, use h:head, h:body
  - Use #{beanName.propertyName} to output values

## Questions?

More info:
http://www.coreservlets.com/JSF-Tutorial/jsf2/ – JSF 2.2 tutorial
http://www.coreservlets.com/JSF-Tutorial/primefaces/ – PrimeFaces tutorial
http://courses.coreservlets.com/jsf-training.html – Customized JSF and PrimeFaces training courses
http://coreservlets.com/ – JSF 2, PrimeFaces, Java 7 or 8, Ajax, jQuery, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training