Lecture 16
JSF2: Programming Basics

Lecture Agenda

1. What is JSF?
2. Pros and Cons of JSF
3. New Features in JSF 2.x
4. Simplified flow of control
5. @ManagedBean and default bean names
6. Default mappings for action controller return values
7. Using bean properties to handle request parameters

What is JSF?

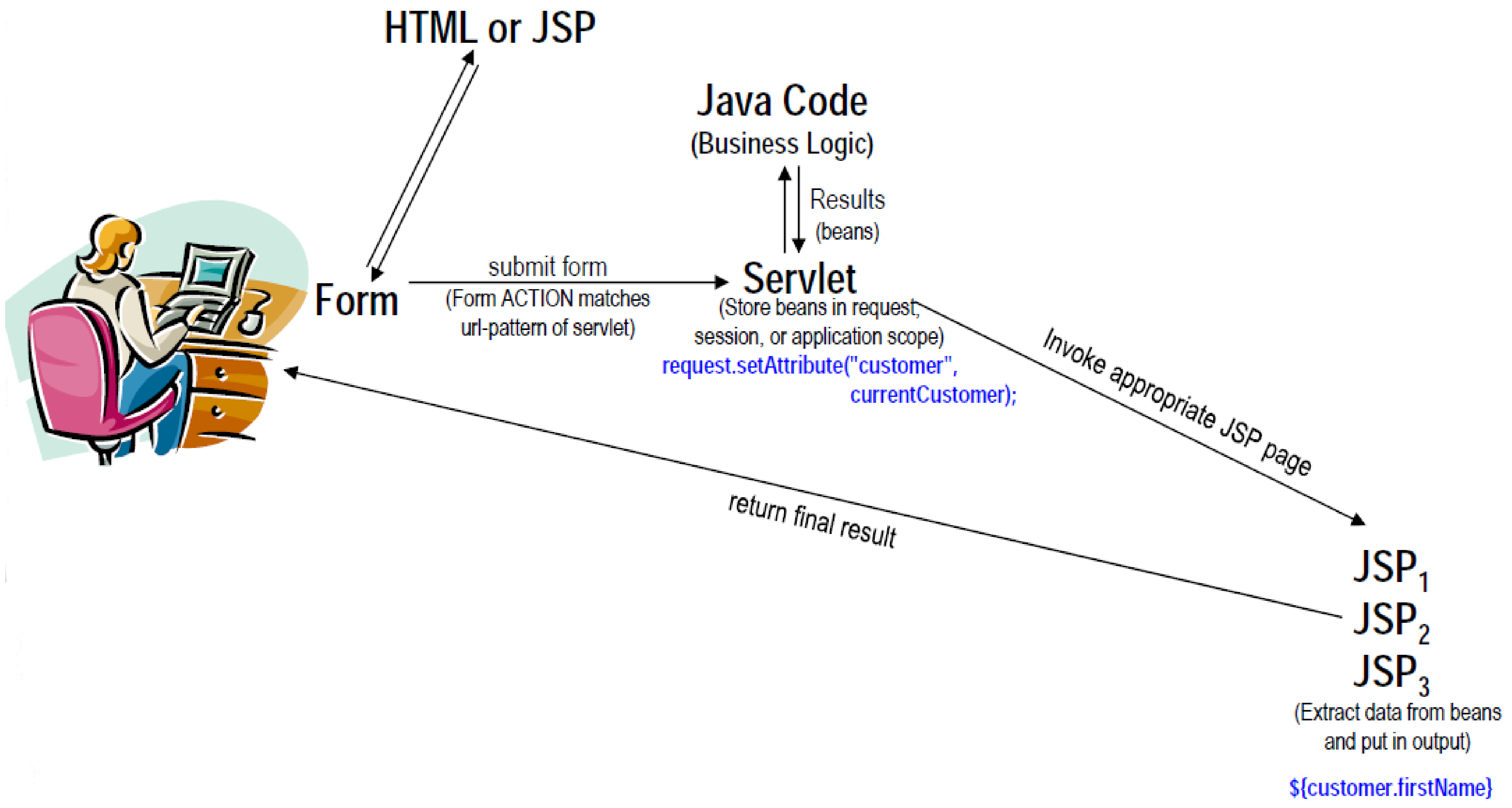
What is JSF?

Introduction and common views of JSF

1. **A set of Web-Based GUI controls and handlers**
   - JSF provides many prebuilt HTML-oriented GUI controls, along with code to handle their events
2. **A device-independent GUI control Framework**
   - JSF can be used to generate graphics in formats other than HTML, using protocols other than HTTP.
3. **An MVC-Based Web Application Framework**
   - JSF can be viewed as an MVC framework for building HTML forms, validating their values, invoking business logic and displaying results.

MVC Review

A Quick Review of MVC



Applying MVC

Example: Bank Account Balances

1. **Bean**
   - BankCustomer
2. **Business Logic**
   - BankCustomerLookup
3. **Servlet populates bean and forwards to appropriate JSP page**
   - Reads customer ID, calls BankCustomerLookup's data-access code to obtain BankCustomer
   - Uses current balance to decide on appropriate result page
4. **JSP pages to display results**
   - Negative balance: warning page
   - Regular balance: standard page
   - High Balance: page with advertisements added
   - Unknown customer ID: error page

Bank Account Balances: Servlet Code

```
public class ShowBalance extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    BankCustomer currentCustomer =
      BankCustomerLookup.getCustomer
                        (request.getParameter("id"));
    request.setAttribute("customer", currentCustomer);
    String address;
    if (currentCustomer == null) {
      address =
        "/WEB-INF/bank-account/UnknownCustomer.jsp";
    } else if (currentCustomer.getBalance() < 0) {
      address =
        "/WEB-INF/bank-account/NegativeBalance.jsp";
    } ...
    RequestDispatcher dispatcher =
      request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
```

Bank Account Balances: Bean

```
public class BankCustomer {
  private final String id, firstName, lastName;
  private final double balance;

  public BankCustomer(String id,
                      String firstName,
                      String lastName,
                      double balance) {
    this.id = id;
    this.firstName = firstName;
    this.lastName = lastName;
    this.balance = balance;
  }

  // Getters for four instance variables. No setters.

  public double getBalanceNoSign() {
    return(Math.abs(balance));
  }
}
```

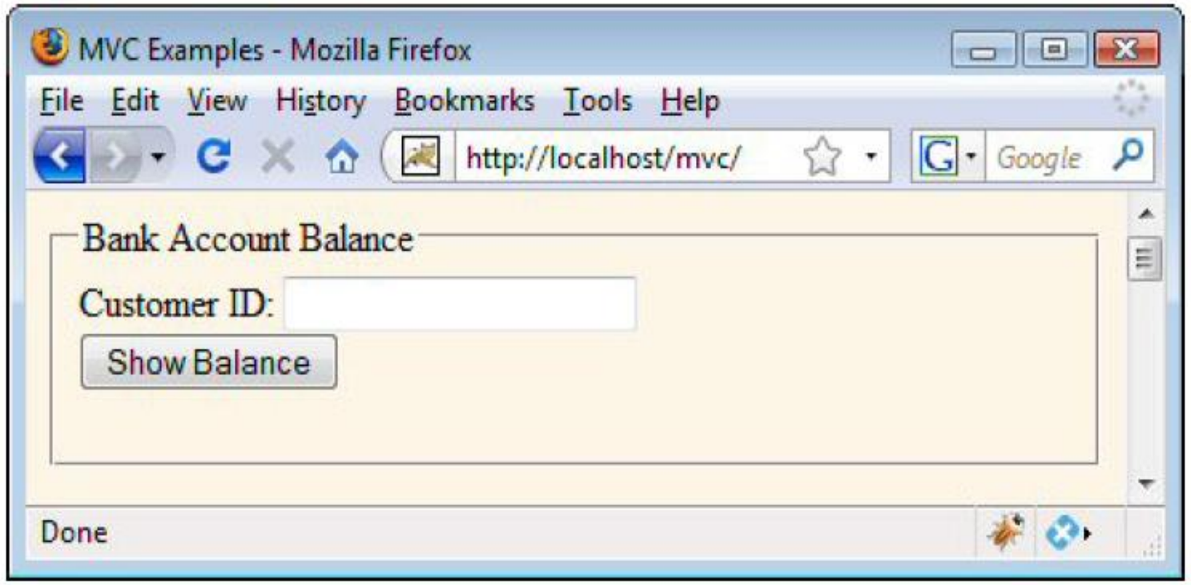
Bank Account Balances: Business Logic

```
public class BankCustomerLookup {
  private static Map<String,BankCustomer> customers;

  static {
    // Populate Map with some sample customers
  }


  …


  public static BankCustomer getCustomer(String id) {
    return(customers.get(id));
  }
}
```

Bank Account Balances: Input Form

```
…
<fieldset>
  <legend>Bank Account Balance</legend>
  <form action="  show-balance">
    Customer ID: <input type="text" name="id"><br>
    <input type="submit" value="Show Balance">
  </form>
</fieldset>
…
```

servlet



Bank Account Balances: JSP Code

```
…
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">
      We Know Where You Live!</TABLE>
<P>
<IMG SRC="/bank-support/Club.gif" ALIGN="LEFT">
Watch out, ${customer.firstName},
we know where you live.
<P>
Pay us the $${customer.balanceNoSign}
you owe us before it is too late!
</BODY></HTML>
```

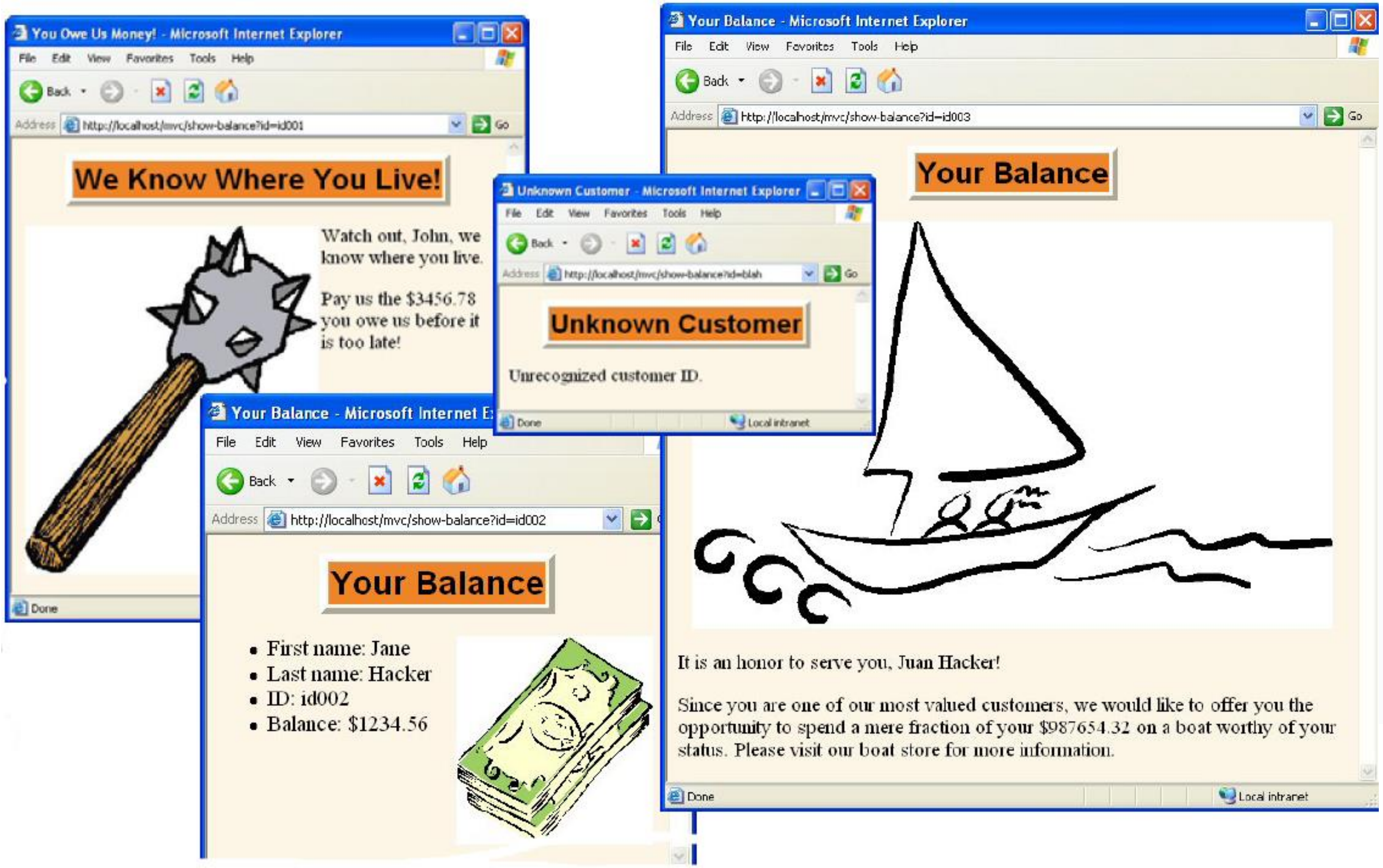
Bank Account Balances: web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" ...>
  <!-- Use the URL http://host/app/show-balance instead of
       http://host/app/servlet/coreservlets.ShowBalance -->
   <servlet>
    <servlet-name>ShowBalance</servlet-name>
    <servlet-class>coreservlets.ShowBalance</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ShowBalance</servlet-name>
    <url-pattern>/show-balance</url-pattern>
  </servlet-mapping>
...
</web-app>
```

Define Servlet

Define Servlet Mapping

Bank Account Balances: Results


You Owe Us Money! - Microsoft Internet Explorer
http://localhost/mvc/show-balance?id=id001
Unknown Customer - Microsoft Internet Explorer
http://localhost/mvc/show-balance?id=blah
Your Balance - Microsoft Internet Explorer
http://localhost/mvc/show-balance?id=id002
Your Balance - Microsoft Internet Explorer
http://localhost/mvc/show-balance?id=id003

We Know Where You Live!

Watch out, John, we know where you live.

Pay us the $3456.78 you owe us before it is too late!

Unknown Customer

Unrecognized customer ID.

Your Balance

- First name: Jane
- Last name: Hacker
- ID: id002
- Balance: $1234.56

Your Balance

It is an honor to serve you, Juan Hacker!

Since you are one of our most valued customers, we would like to offer you the opportunity to spend a mere fraction of your $987654.32 on a boat worthy of your status. Please visit our boat store for more information.

Advantages of JSF

Advantages of JSF

Advantages of JSF vs MVC RequestDispatcher

1. **Custom GUI Controls**
   - JSF provides a set of APIs associated custom tags to create HTML forms that have complex interfaces.
2. **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.
3. **Managed Beans**
   - JSF greatly simplifies parameter (param) processing
4. **Integrated Ajax support**
   - You can use Ajax without explicit javascript programming (using very simple tags)
5. **From 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 are in an improper format, the form can be automatically redisplayed with error messages and with the previously entered values maintained.

Advantages of JSF Continued …

Advantages of JSF vs MVC RequestDispatcher

6. **Page Templating**
   - JSF has a full-fledged page templating system that lets you build pages that share layout or content.
7. **Centralized file-based configuration**
   - Rather than hard-coding information into Java Programs, many JSF values are represented in XML or property files. This loose coupling means that many changes can be made without modifying or recompiling Java code, and that wholesale changes can be made by editing a single file.
8. **Consistent Approach**
   - JSF encourages consistent use of MVC throughout your application.

Disadvantages of JSF

Disadvantages of JSF vs MVC RequestDispatcher

1. **Bigger Learning Curve**
   - To use MVC with standard RequestDispatcher, you need to be comfortable with standard JSP and servlet APIs.
   - To use MVC with JSF, you have to be comfortable with servlet API and a large and elaborate framework that is almost equal in size to the code system.
2. **Worse Documentation**
   - Compared to the standard servlet and JSP API's, JSF has fewer online resources and many first-time users find the online JSF documentation confusing and poorly organized.
3. **Less Transparent**
   - With JSF applications, there is a lot more going on behind the scenes than with normal Java-based Web applications. As a result, JSF applications are: Harder to understand, Harder to benchmark and Harder to optimize
4. **Rigid Approach**
   - JSF encourage a consistent approach to MVC that can make it difficult to use new/other approaches

New Features in JSF 2.x

Features in JSF 2.x

New Features in JSF 2.x

1. **JSF is the official Java EE library for Web Applications**
   - JSF 2 (with a rich component library like PrimeFaces or RichFaces) is the most popular choice in practice.
2. **JSF 2.x adds many new features**
   - Smart defaults
   - Annotations as alternatives to face-config.xml entries
   - Integrated AJAX support
   - Facelets (.xhtml files) instead of JSP
   - Ability to bookmark results pages

Main JSF 2.x Implementations

Main Providers

1. **Oracle Mojarra**
   - Main page: https://javaserverfaces.java.net/
   - Runs in any server supporting servlets 3.x or later
   - Integrated in Glassfish 4
2. **Apache MyFaces**
   - Main page: http://myfaces.apache.org/core22/
   - Runs in any server supporting servlets 3.x or later
3. **Any Java EE 7 server**
   - JSF 2.2 is official built-in part of Java EE 7
   - JBoss 8, Glassfish 4
   - WebLogic, WebSphere 8 etc …

JSF Setup

Requirements for Running JSF 2.x

Main Providers

1. **Java**
   - Java 7 or Java 8 preferred, but java 6 is technically legal
   - Java EE 7 servers run on top of Java 7, but **Java 8** remains the best option
2. **Server**
   - Servlet engine supporting 3.x (by including JSF jar file)
   - Tomcat 8, Glassfish 4, Jboss 8, WebSphere and WebLogic
3. **IDE**
   - Eclipse again is highly recommended
   - You can however also use, NetBeans and IntelliJ IDEA

Software Required Summary

Side by Side Summary

To run on Tomcat

1. Install Java 8
2. Install IDE / Eclipse
3. Download and Install Tomcat 8
4. Get JSF 2.2 Jar File
   - Download Oracle (Mojarra) or Apache (MyFaces)
5. web.xml, face-config.xml
   1. Required entries (shown in later lecture)

To run on Java EE 7

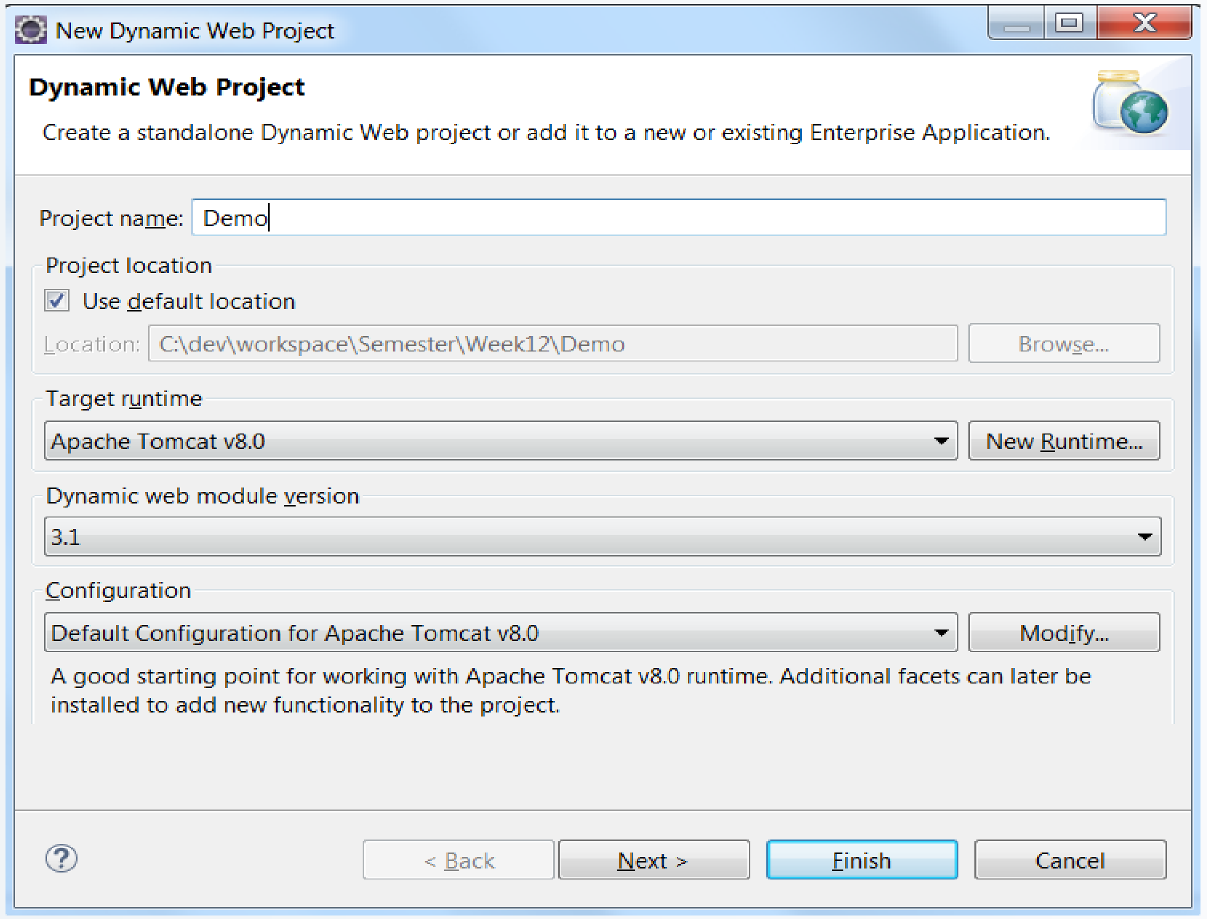
1. Install Java 8
2. Install IDE/ Eclipse
3. Download Glassfish 4
   - Or any server supporting Java EE 7
4. No extra JAR files needed
   - Java EE 7 has built-in support for JSF 2.2
5. web.xml, face-config.xml
   - Required entries (shown in later lecture)

Making JSF 2.2 Project with Eclipse Wizard

Steps Required to Create JSF Project

JSF With Eclipse

Step1: Create New Dynamic Web Project


New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: Demo

Project location

☑ Use default location

Location: C:\dev\workspace\Semester\Week12\Demo

Target runtime: Apache Tomcat v8.0

Dynamic web module version: 3.1

Configuration: Default Configuration for Apache Tomcat v8.0

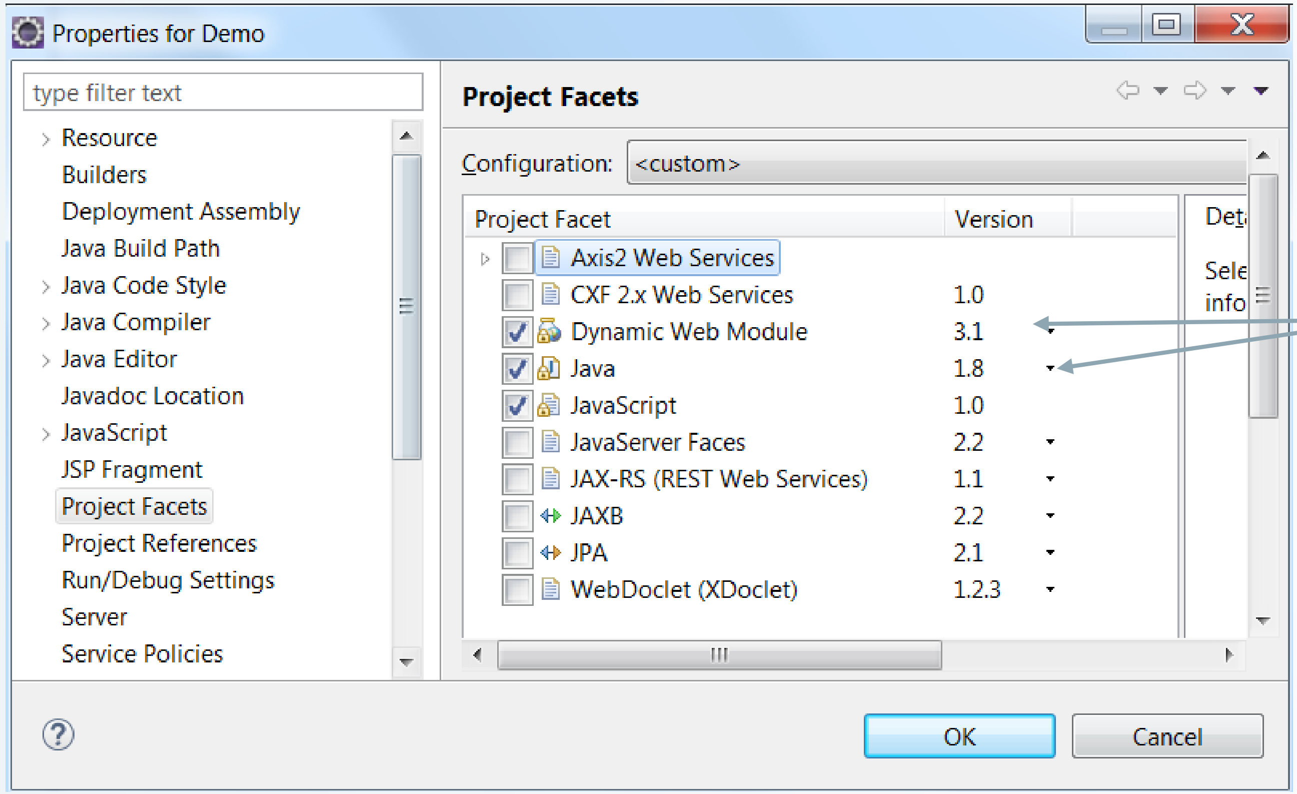
A good starting point for working with Apache Tomcat v8.0 runtime. Additional facets can later be installed to add new functionality to the project.

Steps Required to Create JSF Project

JSF With Eclipse

Step 2: Right Click Project → Properties

Step 3: In Properties, Select → Project Facet


Properties for Demo

type filter text

- Resource
- Builders
- Deployment Assembly
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- JavaScript
- JSP Fragment
- Project Facets
- Project References
- Run/Debug Settings
- Server
- Service Policies

Project Facets

Configuration: <custom>

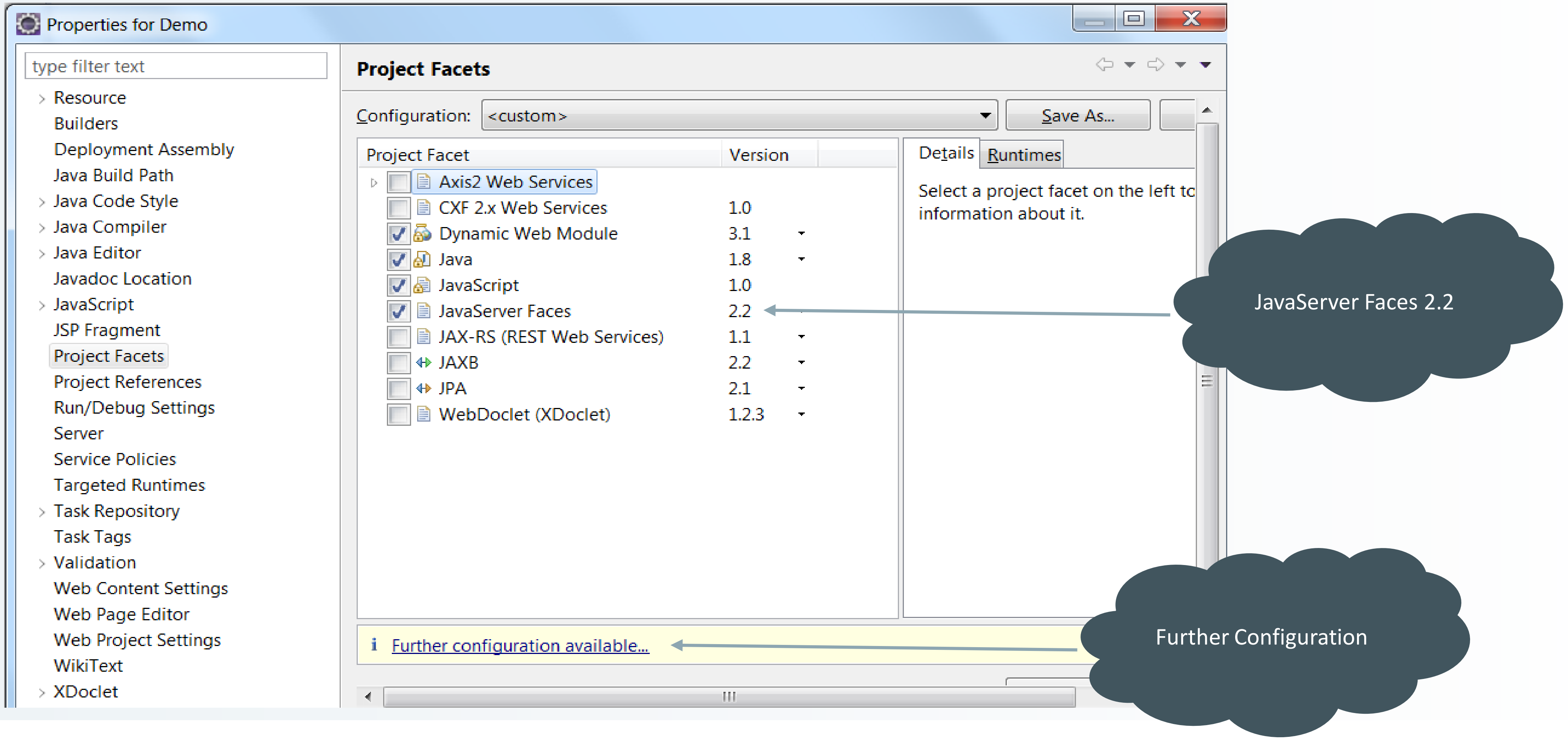
Project Facet	Version
☐ Axis2 Web Services	
☐ CXF 2.x Web Services	1.0
✓ Dynamic Web Module	3.1
✓ Java	1.8
✓ JavaScript	1.0
☐ JavaServer Faces	2.2
☐ JAX-RS (REST Web Services)	1.1
☐ JAXB	2.2
☐ JPA	2.1
☐ WebDoclet (XDoclet)	1.2.3

Validate the version of Java is **1.8** and Dynamic Web Module is **3.1**

Steps Required to Create JSF Project

JSF With Eclipse

Step 3: Tick JavaServer Faces and select at version 2.2



Properties for Demo

type filter text

- Resource
- Builders
- Deployment Assembly
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- JavaScript
- JSP Fragment
- Project Facets
- Project References
- Run/Debug Settings
- Server
- Service Policies
- Targeted Runtimes
- Task Repository
- Task Tags
- Validation
- Web Content Settings
- Web Page Editor
- Web Project Settings
- WikiText
- XDoclet

Project Facets

Configuration: <custom>

	Project Facet	Version
☐	Axis2 Web Services	
☐	CXF 2.x Web Services	1.0
✓	Dynamic Web Module	3.1
✓	Java	1.8
✓	JavaScript	1.0
✓	JavaServer Faces	2.2
☐	JAX-RS (REST Web Services)	1.1
☐	JAXB	2.2
☐	JPA	2.1
☐	WebDoclet (XDoclet)	1.2.3

Details | Runtimes

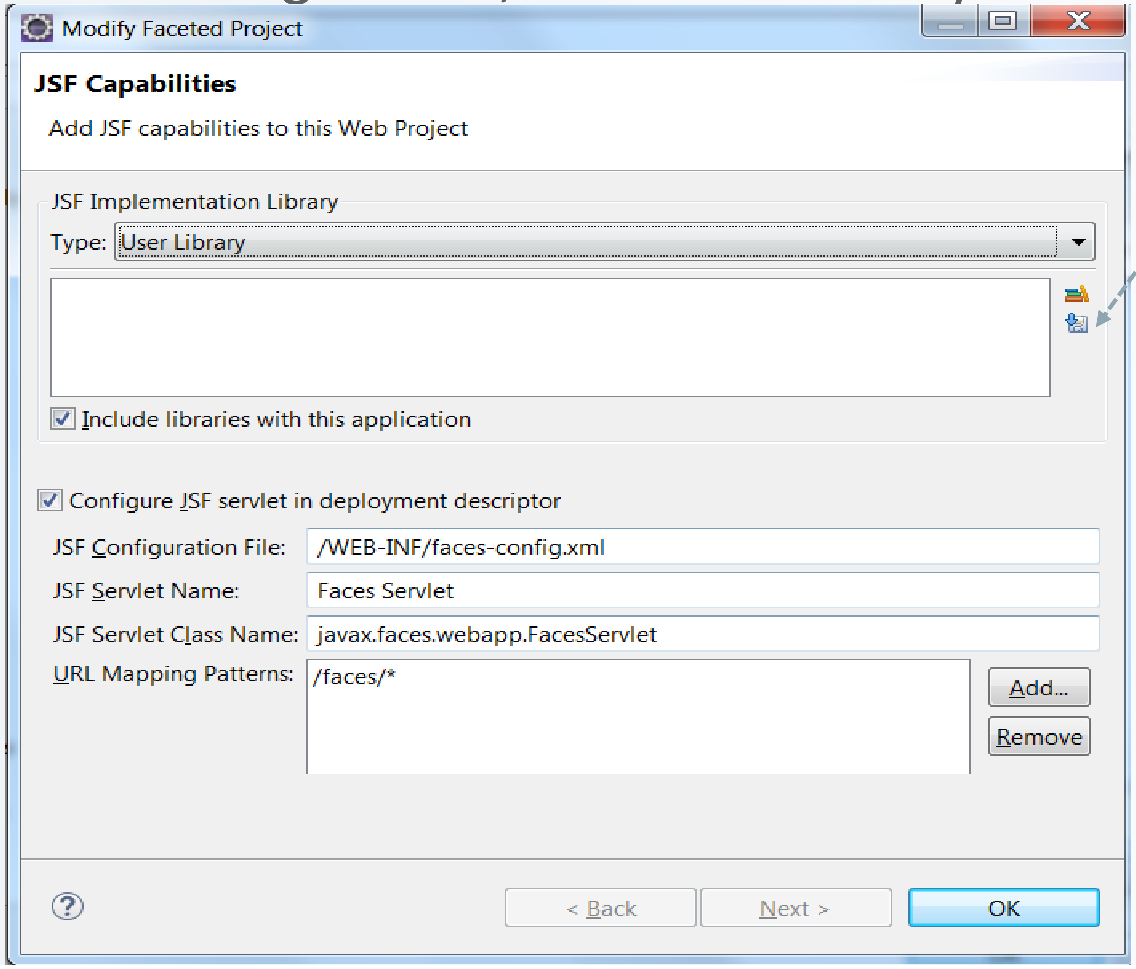
Select a project facet on the left to information about it.

Further configuration available...

Steps Required to Create JSF Project

JSF With Eclipse

Step 4: Click on Further Configuration, Download library



JSF Capabilities

Add JSF capabilities to this Web Project

JSF Implementation Library

Type: User Library

[✓] Include libraries with this application

[✓] Configure JSF servlet in deployment descriptor

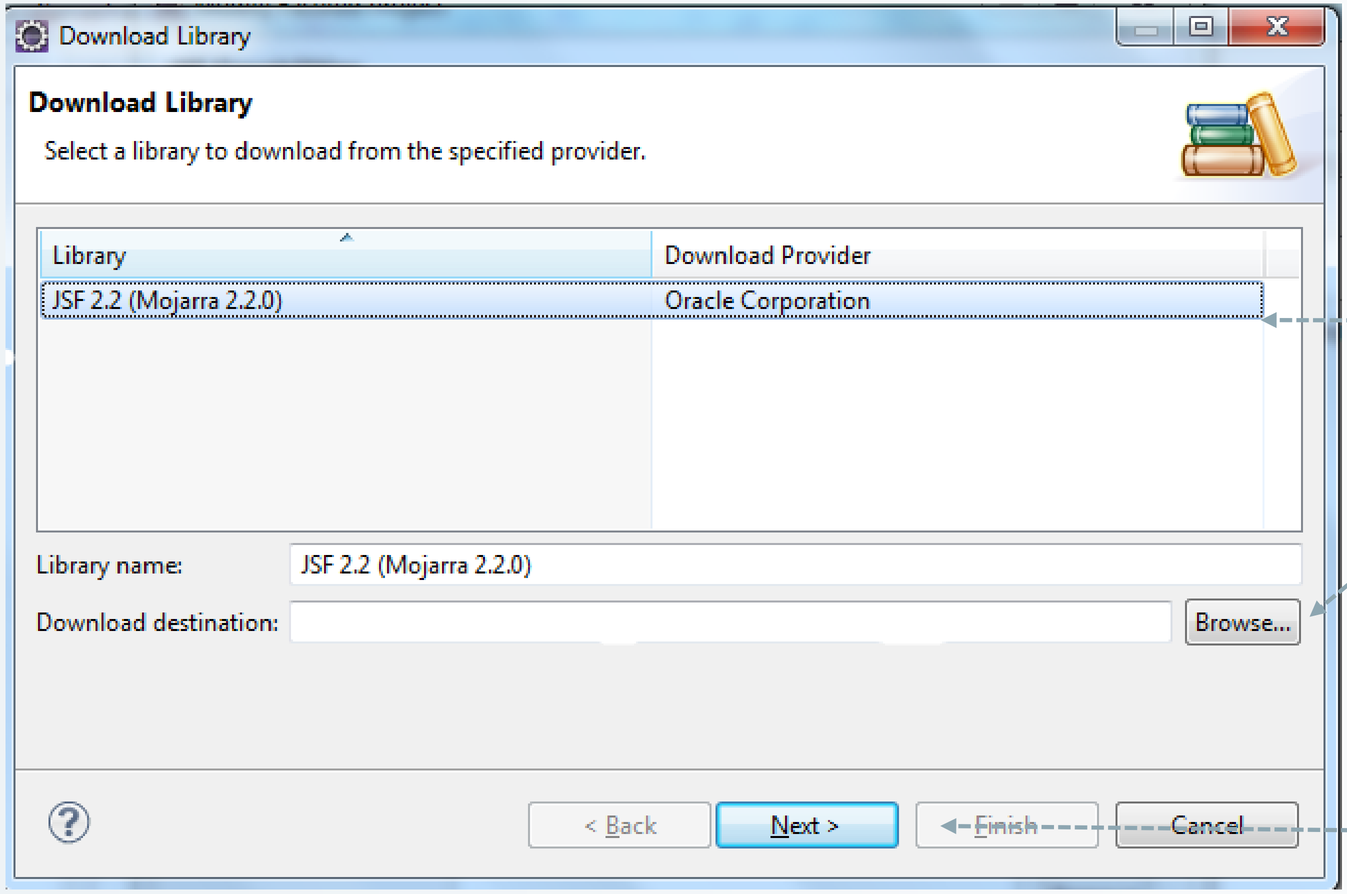
JSF Configuration File:	/WEB-INF/faces-config.xml
JSF Servlet Name:	Faces Servlet
JSF Servlet Class Name:	javax.faces.webapp.FacesServlet
URL Mapping Patterns:	/faces/*

Click on Download Library

Steps Required to Create JSF Project

JSF With Eclipse

Step 5: Tick/Select Mojarra 2.2.x from (Oracle JSF Implementation)


Download Library

Download Library

Select a library to download from the specified provider.

Library	Download Provider
JSF 2.2 (Mojarra 2.2.0)	Oracle Corporation

Library name: JSF 2.2 (Mojarra 2.2.0)

Download destination:

Select Mojarra Library. Specify desired location.

Click Next Button

Steps Required to Create JSF Project

JSF With Eclipse

Step 6: Click okay to Close Dialog , Click okay to close project facet dialog

By the end of the above steps, you are ready for using JavaServer Faces in your project.

Jab

JSF Setup Summary

JSF Setup Summary

Main Providers

- **JAR files**
  - JSF 2.2 JAR file required
    - Omit this step in full Java EE compliant container (Glassfish 4, Jboss 7/8, other Java EE servers)
- **face-config.xml**
  - For this entire lecture: empty body (start/end tags only)
    - This lecture use Java-based annotations and default mappings of action controller values to results pages.

- **web.xml**
  - Must have a url-pattern for *.jsf (or other pattern you choose)
  - Usually sets **PROJECT_STAGE** to development
- **Accessing files (some-page.xhml)**
  - Use URL **some-page.jsf** (matches url-pattern from web.xml)

face-config.xml

Example

```
<?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 is has legal start and end tags should you choose to use it later

There will be no content inside the tags for this lecture. All examples in this lecture use default bean names, and default result pages

web.xml (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>
```

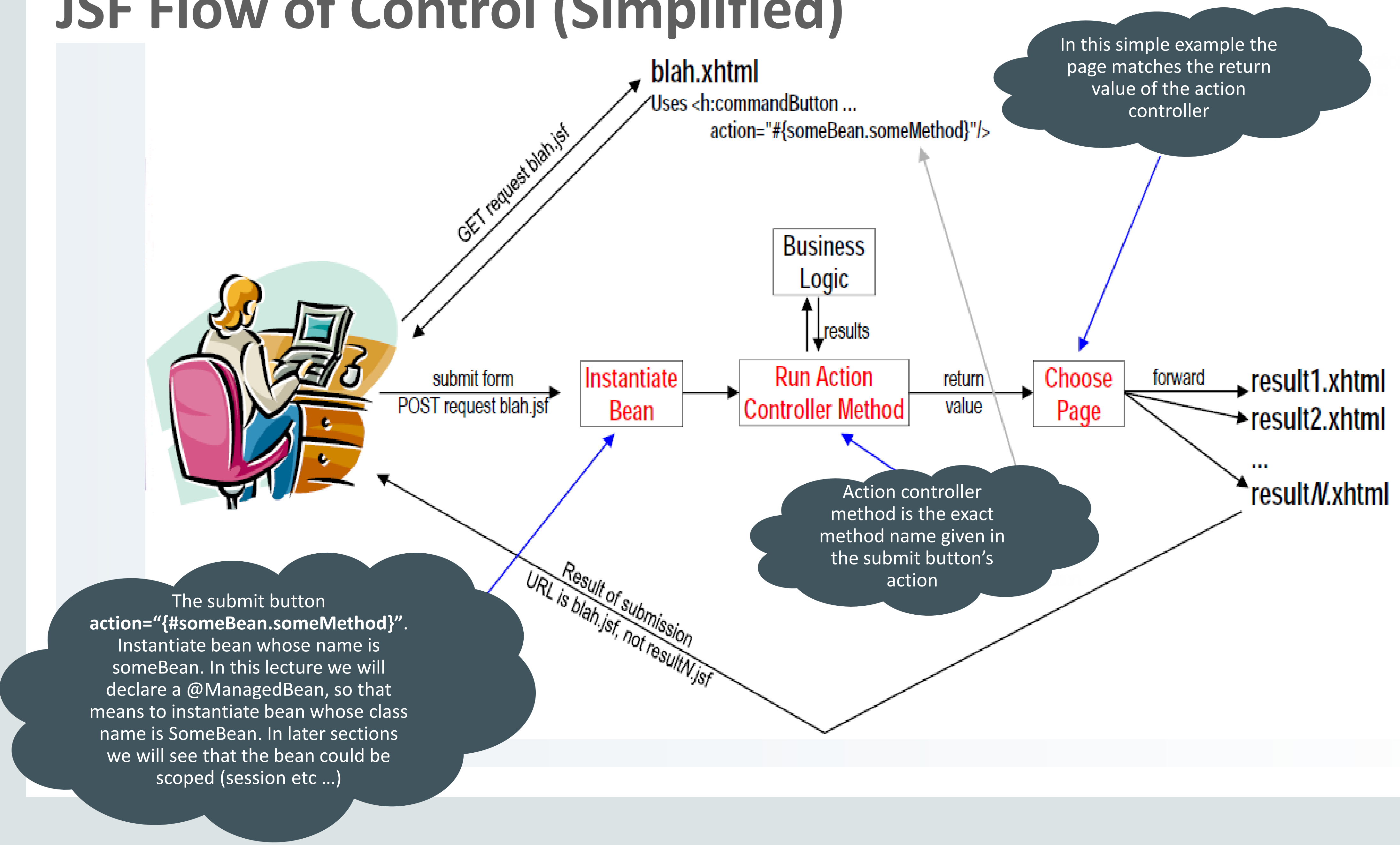
The real file is **blah.xhtml** but the URL is **blah.jsf**

Means that you get extra debugging support

Home page of application

Basic Structure of JSF 2 Applications

JSF Flow of Control (Simplified)



In this simple example the page matches the return value of the action controller

Action controller method is the exact method name given in the submit button's action

The submit button **action="{#someBean.someMethod}"**. Instantiate bean whose name is someBean. In this lecture we will declare a @ManagedBean, so that means to instantiate bean whose class name is SomeBean. In later sections we will see that the bean could be scoped (session etc …)

Basic Structure of Facelets Page

```
<!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>
```

Remember that then URL does not match the real filename: You use blah.xhtml for the file, but blah.jsf for the URLs 9or whatever pattern you described in web.xml)

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 annotated with @ManagedBean

Managed Beans are POJO’s (they implement no special interfaces, and most methods have no JSF specific argument or structure).

Note: They have pairs of Setters/Getters

They have a common action controller method that takes no arguments and returns a String.

@ManagedBean Basics

@ManagedBean Basics

Main Points

- **@ManagedBean annotation**

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

  - You refer to bean with #{someName.method}, where bean name is class name (minus package) with first letter changed to lowercase.
  - **Request** scope by default

- **Return values of action controller method**
  - If action controller returns “**foo**” and there are **no explicit mappings** in **faces-config.xml**, then results page is **foo.xhtml**.
  - Where foo.xhtml is in the same folder as the page that contained form.

Practical Example

Practical Example
JSF 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, 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>
```

Note the jsf import

This means that when you press the button, JSF instantiates bean whose name is navigator and then runs the choosePage() method. The name of the 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));
  }
}
```

Declared as ManagedBean

Since no name is given, name is class name (first letter lowercase ex navigator). You can also do **@ManagedBean(name="someName")**

Since there is no scope defined, it is request scoped. You can also use annotation like @SessionScoped etc …

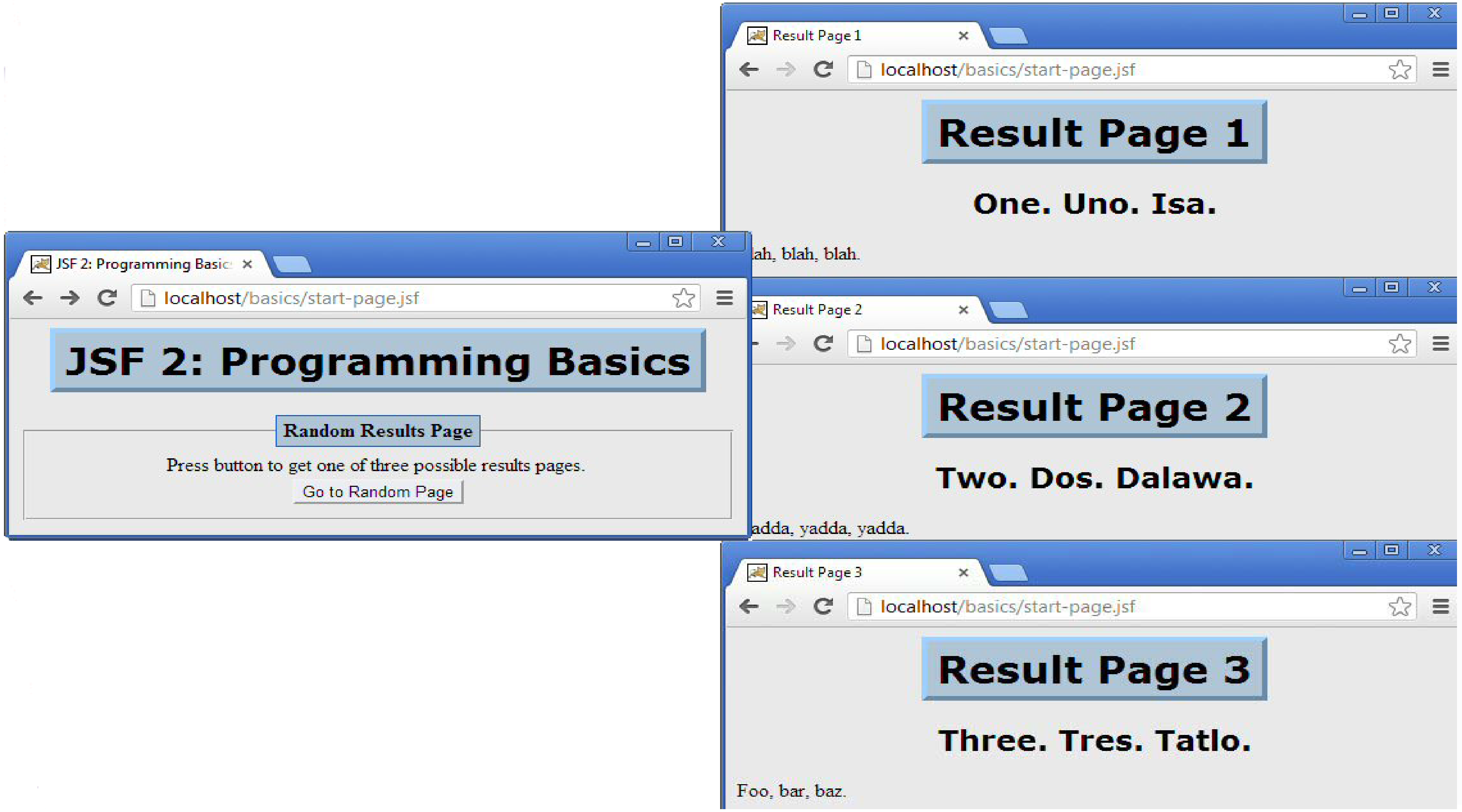
Since there are no explicit rules defined in our face-config.xml, these return values correspond to page1.xhtml, page2.xhtml, page3.xhtml (same folder as the form)

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>
```

Even though there is no dynamic code, it is good practice to include h:head, h:body

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

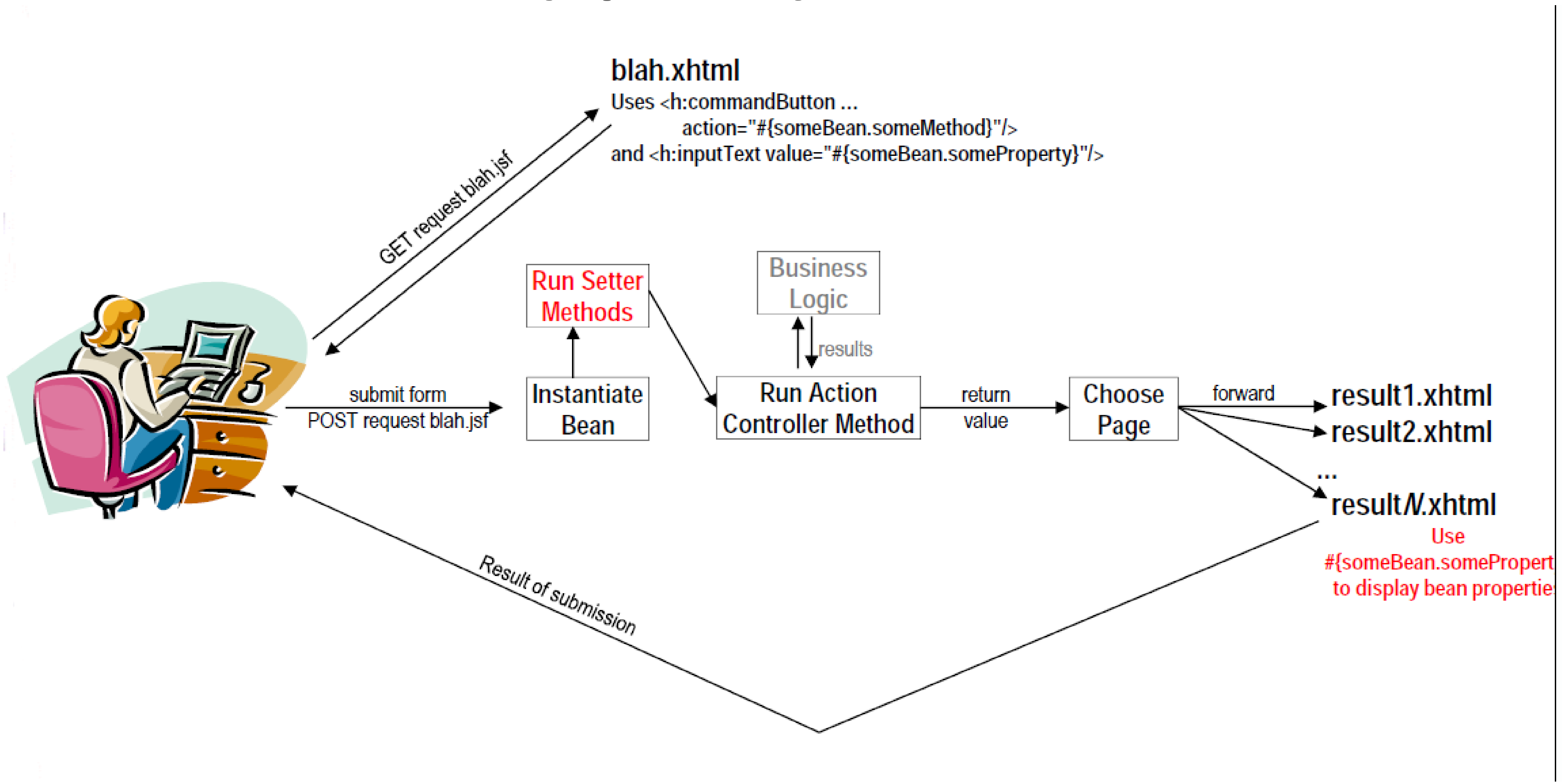
Result Page 3

Three. Tres. Tatlo.

Foo, bar, baz.

Using Beans to handle Request Parameters

JSF Flow of Control (Updated)



Main Points

JSF Request Parameter Main Points

- **Input values correspond to bean properties**
  - <h:inputText value="#{someBean.someProperty}"/>
    - When form is submitted, takes value in text field and passes it to setSomeProperty()
      - Validation and type conversion (if any) is performed.
    - When form is displayed, calls getSomeProperty(). If value is **other** than null or empty String, puts value in field.
- **Beans are request scoped by default**
  - Bean is instantiated **twice**, once when form is initially displayed, then again when the form is submitted.
- **Can use #{bean.SomeProperty} directly in output**
  - Means to output result of getSomeProperty()
    - Instead of <h:outputText value="#{bean.someProperty}"/>

Using Beans to handle Request Parameters Example

Example
JSF Request Parameter Main Points

- **Purpose**
  - 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
- **Features Required**
  - Bean
    - Properties corresponding to request parameters
  - Input form
    - `<h:inputText value="{languageForm.language}">`
  - Results Page
    - `#{languageForm.language}` (for warning page)

choose-language.xhml

Input form

```
<!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

Afterward showChoice method is called to determine the results page

The value of h:inputText actually plays a dual role. When the form is first displayed, languageForm is instantiated and getLanguage is called. If the value is non-empty, that result is the initial value of text field. Otherwise text field is initially empty. When the form is submitted, languageForm is re-instantiated and the value in textfield is passed to setLangauge.

LanguageForm.java
ManagedBean

```
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 automatically be called by JSF when form is submitted

LanguageForm.java continued …

ManagedBean

```
  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 name

missing-language.xhtml
Results Page

```
<!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

Results Page

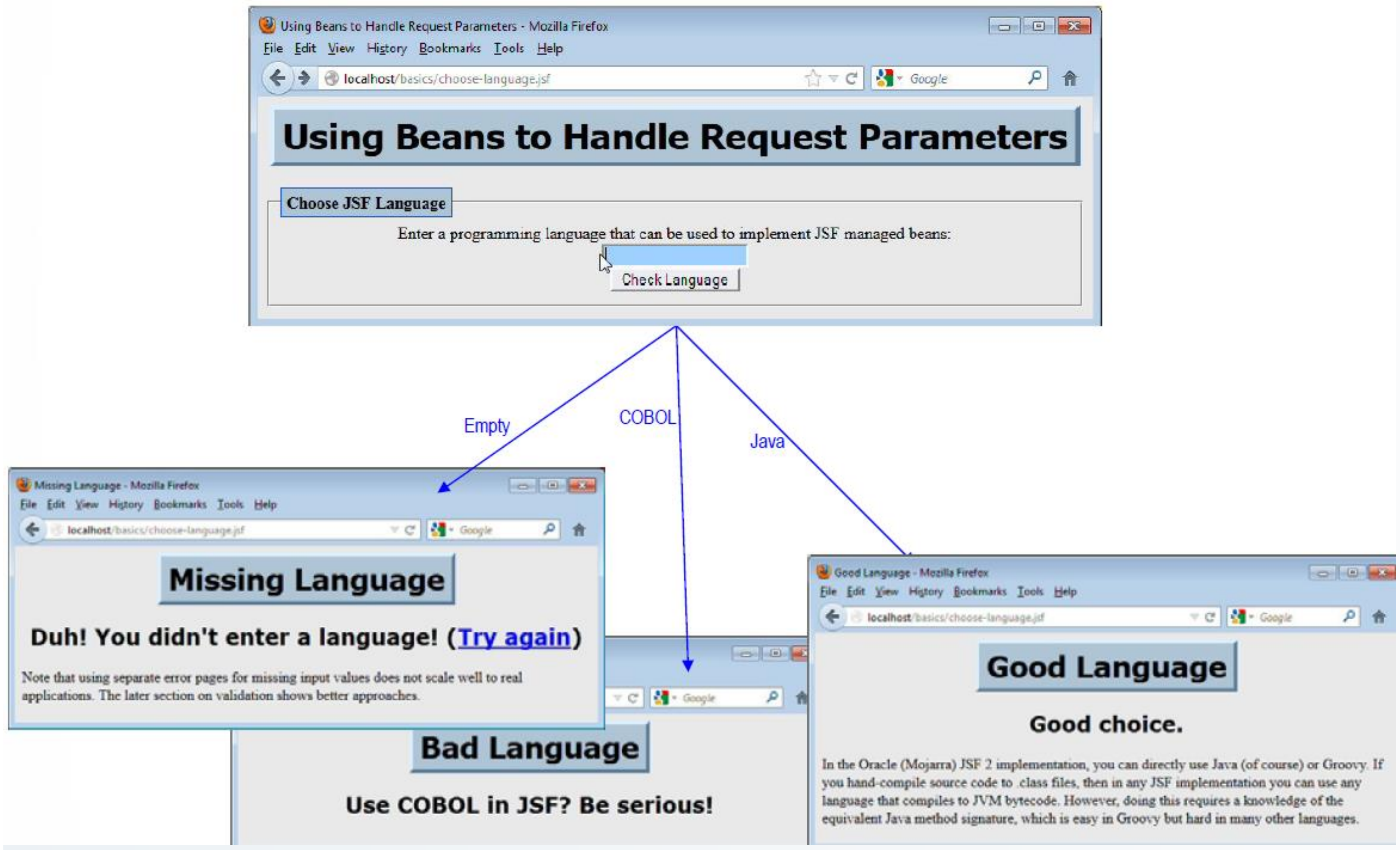
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
…
</h:head>
<h:body>

<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

good-language.xhtml
Results Page

```
<!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.

Summary

- **Input pages, forms, Output/Results Pages (facelets pages)**
  - Is the JSF view handler. Before JSF, JSP use to be the default technology view handler
  - Declare h:namespace, use h:head, h:body, h:form (for forms)
- **Java Code: managed beans**
  - Declare with @ManagedBean
    - Bean name is the class name with the first letter in lower case.
    - Getter and Setter methods for each input element defined
      - Example: <h:inputText value=“#{beanName.propertyName}” />
    - Action controller method
      - Form: <h: commandButton action=“beanName.methodName”>
      - Return values become base names of result page(s)
- Results pages
  - Declare h:namespace, use h:head, h:body
  - Use #{beanName.propertyName} to output values

Questions?