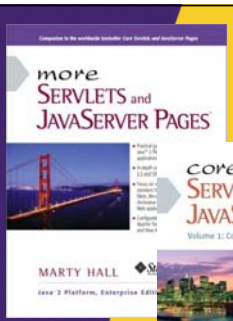
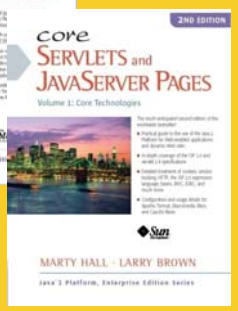
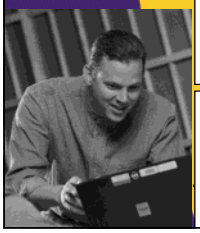
The JSF 2 Expression Language

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

- **Motivating use of the expression language**
  - Comparing to the JSF 1.*x* and JSP 2.0 ELs
- **Simplified testing of EL capabilities**
- **Accessing bean properties**
  - Direct
  - Nested
- **Submitting bean properties**
  - Expressions in output values
  - Expressions in submission values
  - Expressions for action controllers
- **Accessing collection elements**
- **Using implicit objects and operators**
- **Conditionally rendering output**
- **Passing arguments to methods**

Overview

The Expression Language

- **JSP scripting not supported in facelets**
  - So, you need a way to indirectly invoke Java
- **Quick 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

Advantages of the Expression Language (Very Important)

- **Shorthand notation for bean properties**
  - To reference the result of the getCompanyName method of a managed bean named company, you use #{company.companyName}.
  - To reference the firstName property of the president property of a managed bean named company, you use #{company.president.firstName}.
- **Simple access to collection elements**
  - To reference an element of an array, List, or Map, you use #{someBean.someProperty[indexOrKey]}.
    - E.g., #{person.friends[2]}

Advantages of the EL (Moderately Important)

- **A small but useful set of simple operators**
  - To manipulate objects within EL expressions, you can use any of several arithmetic, relational, logical, or empty-testing operators.
- **Conditional output**
  - To choose among output options:
    - #{test ? option1 : option2}
    - <h:someElement … rendered="#{test}"/>
    - <ui:fragment rendered="…">…</ui:fragment>
      - We will give very brief examples in this tutorial section. The later section on looping with ui:repeat will give more details.

Advantages of the EL (Less Important)

- **Predefined variables (implicit objects)**
  - To access request params, cookies, HTTP headers, and other standard types of request data, you can use one of several predefined implicit objects.
- **Passing arguments**
  - Version 2.1 of the EL lets you pass arbitrary arguments to methods. Works only in Java EE 6 or other servers that support EL 2.1. *Not part of JSF 2 itself.*
    - E.g, works in Tomcat 7 but not Tomcat 6, even though JSF 2 works in both.
- **Empty values instead of error messages**
  - In most cases, missing values or NullPointerExceptions result in empty strings, not thrown exceptions.

JSF vs. JSP ELs

Feature	JSF 2.0 EL	JSF 1.*x* EL (with JSP)	JSP 2.0 EL
Format	#{blah} (immediate output values could be accessed with ${blah})	#{blah}	${blah}
Where used	Anywhere in facelets page Eg: #{customer.firstName}	Only in attributes of JSF tags. Eg: <h:outputText value= "#{customer.firstName}"/>	Anywhere in page Eg: ${customer.firstName}
Represents	Output data, later location for submitted data. Eg: <h:inputText value= "#{customer.firstName}"/>	Output data, later location for submitted data. Eg: <h:inputText value= "#{customer.firstName}"/>	Output data. Eg ${customer.firstName}
Where it looks for beans	Request, session, application (etc.) scopes and managed bean defs.	Request, session, application (etc.) scopes and managed bean defs.	Request, session, application scopes.
Declaration type	None needed for simplest usage. xlmns declaration for h:, ui:, f: tags.	@taglib	None needed
Environments	Java EE 6 servers or servlet 2.5 servers with JSF 2.0 JARs.	Java EE 5 servers or servlet 2.4 servers with JSF 1.*x* JARs.	Servlet 2.4+ servers

Simplified Testing of EL Capabilities

Simplifying Testing of the EL

- **JSP**
  - Checks existing scopes (request, session, etc.). If not found, gives up.
- **JSF**
  - Checks existing scopes (request, session, etc.). If not found, looks for managed bean definition of that name (either from @ManagedBean or from faces-config.xml).
- **Implication for testing and experimenting**
  - You can create a simple bean and a simple standalone page to test it. No form, no action controller, no results page. Great for experimenting with EL features.
    - See next page for an example

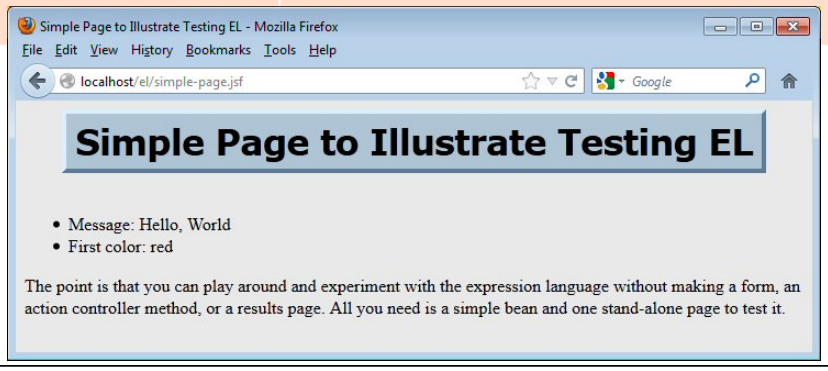
Simplifying Testing of the EL: Example

Bean

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

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

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

Standalone Test Page

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



Outputting Simple Bean Properties

Outputting Simple Bean Properties

- **Format**
  - #{varName.propertyName}
  - <h:outputText value="#{varName.propertyName}" …/>
    - For new JSF 2 code, top version is usually used unless you need some other attribute of h:outputText (e.g. “id”, “rendered”, or “escape”)
- **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());
  }
}
```


Bean Properties Example: Facelets Code

```
<!DOCTYPE …>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head><title>Accessing Simple Bean Properties</title>
<link href="./css/styles.css"
      rel="stylesheet" type="text/css"/>
</h:head>
<h:body>
…
<ul>
  <li>Creation time: #{testBean1.creationTime}</li>
  <li>Greeting: #{testBean1.greeting}</li>
  <li>Random number: #{testBean1.randomNumber}</li>
</ul>
</h:body></html>
```

Bean Properties Example: Result

Accessing Nested Bean Properties

Nested Bean Properties

- **Format**
  - #{var.prop1.prop2.prop3}
  - <h:outputText value="#{var.prop1.prop2.prop3}" …/>
    - Again, use this form only if you need some extra attribute of h:outputText such as “id”, “rendered”, or “escape”
- **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); }

  ...
}
```

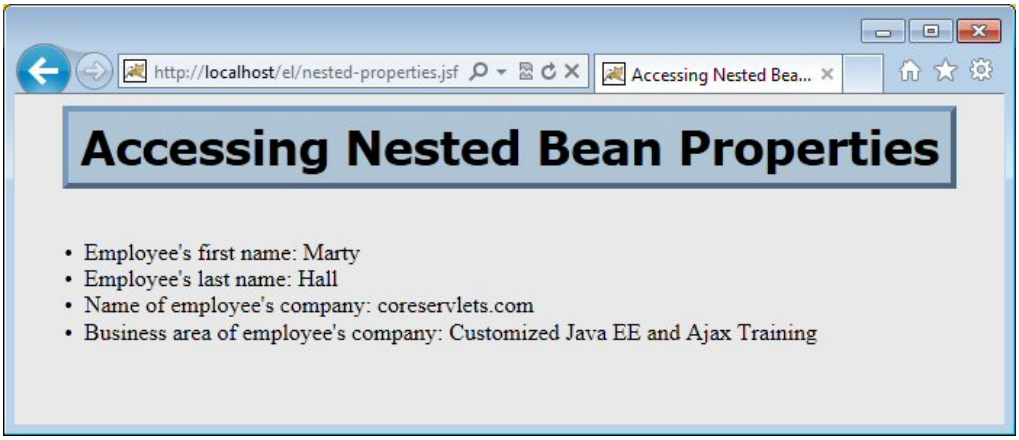
Nested Properties Example: Employee1

```
@ManagedBean
public class Employee1 extends Employee {
  public Employee1() {
    super(new Name("Marty", "Hall"),
          new Company("coreservlets.com",
                      "Customized Java EE and Ajax Training"));
  }
}
```


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



Submitting Bean Properties

Three Uses of #{...}

- **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.

Understanding 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. Discussed later.

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: Facelets Code for Form

```
…
<h:form>
<h:panelGrid columns="2" styleClass="formTable">
Your first name:
<h:inputText value="#{employee1.name.firstName}"/>
Your last name:
<h:inputText value="#{employee1.name.lastName}"/>
Name of your company:
<h:inputText value="#{employee1.company.companyName}"/>
Business area of your company:
<h:inputText value="#{employee1.company.business}"
             size="38"/>
</h:panelGrid>
<h:commandButton value="Process"
                 action="#{employee1.processEmployee}"/>
</h:form>
```

h:panelGrid is a shortcut for making an HTML table. More details in the lecture on validating form data.

Submitting Properties Example: Input Page Initial Result

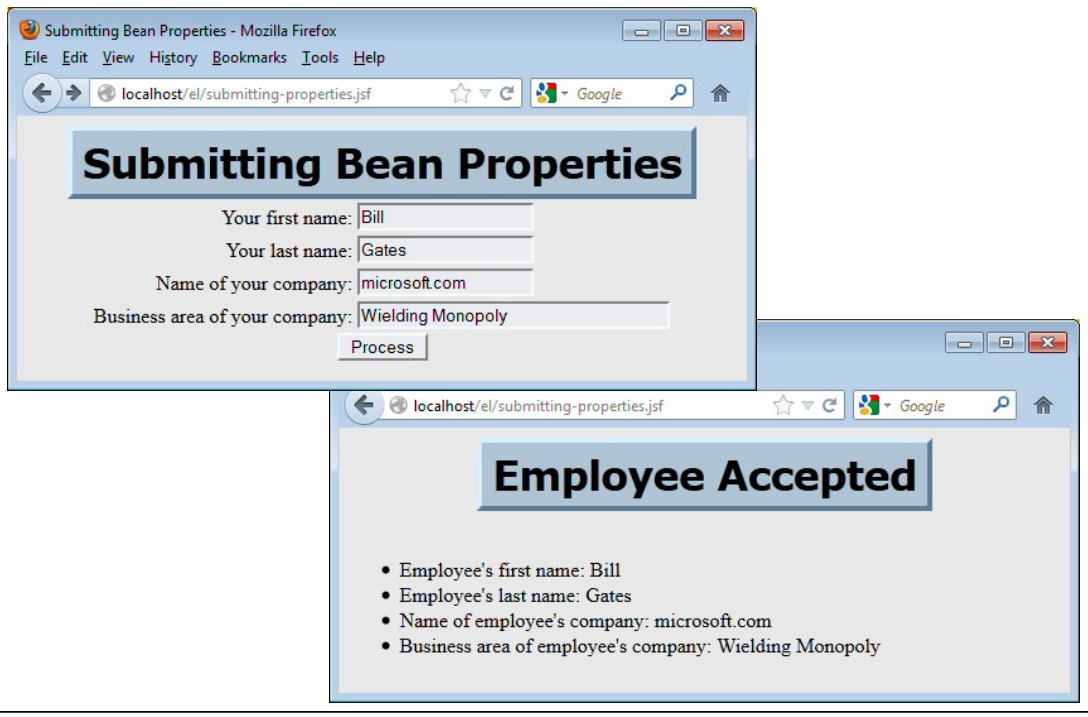
Submitting Properties Example: accepted.xhtml

```
…
<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>
…
```

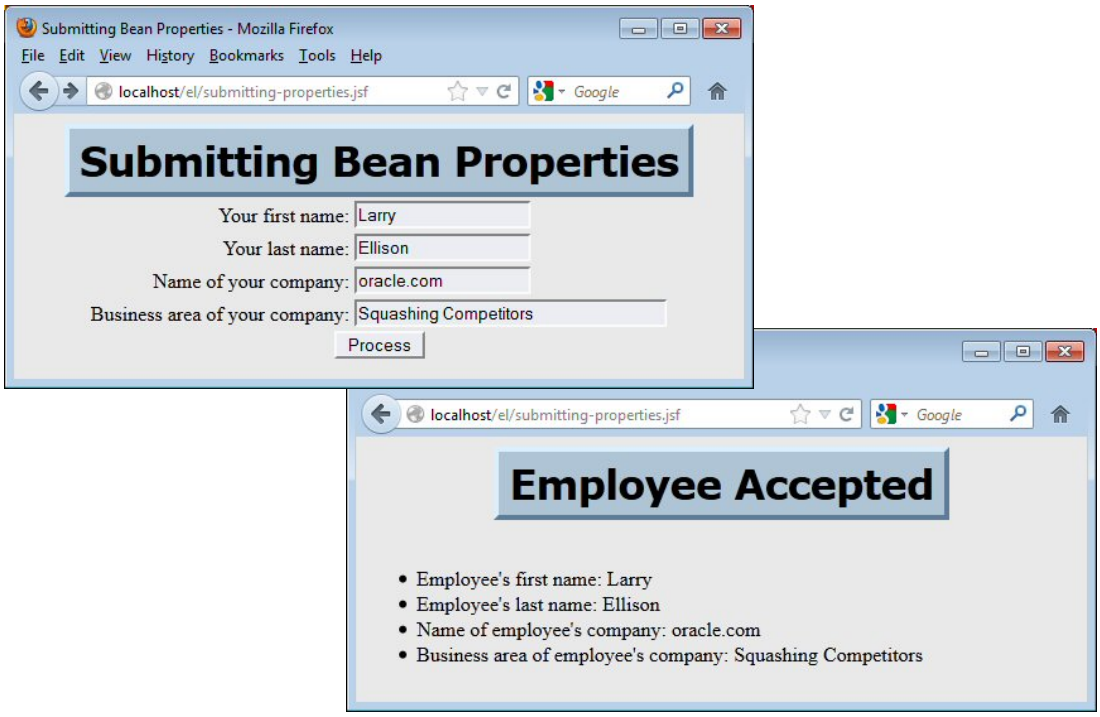
Submitting Properties Example: rejected.xhtml

```
…
<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>
…
```


Submitting Properties Example: Results



Submitting Properties Example: Results (Continued)



Accessing Collections

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
    - See upcoming slides
  - 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 Maps)**
  - #{stateCapitals["maryland"]}
  - #{stateCapitals.maryland}
  - But you can't use this for lists (numbers are not legal Java variables names, so #{listVar.2} is illegal). And not all hash table keys are legal variable names. So, use brackets.

Collections Example: Purchases

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


Collections Example: Purchases (Continued)

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

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

  public Map<String,String> getValuableItems() {
    return(valuableItems);
  }
```

Collections Example: Purchases (Continued)

```
  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);
  }
}
```

Collections Example: using-collections.xhtml

```
...
<h:form>
<ul>
<li><b>Cheap Items</b>
<ol>
<li>
  <h:inputText value="#{purchases.cheapItems[0]}"/>
</li>
<li>
  <h:inputText value="#{purchases.cheapItems[1]}"/>
</li>
<li>
  <h:inputText value="#{purchases.cheapItems[2]}"/>
</li>
</ol></li>
```

This example uses explicit indices. See the tutorial section on looping to see how to redo this example with ui:repeat and a variable for the index.

Collections Example: using-collections.xhtml (Continued)

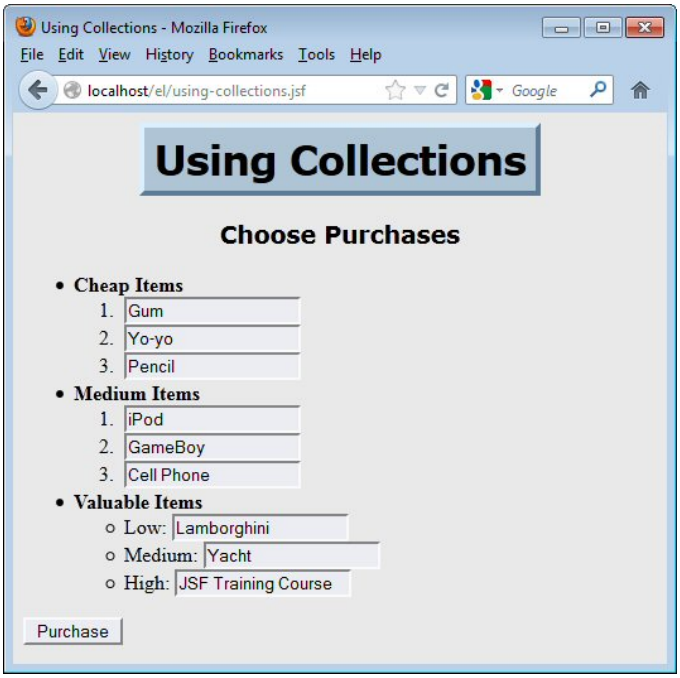
```
<li><b>Medium Items</b>
<ol>
<li>
  <h:inputText value="#{purchases.mediumItems[0]}"/>
</li>
<li>
  <h:inputText value="#{purchases.mediumItems[1]}"/>
</li>
<li>
  <h:inputText value="#{purchases.mediumItems[2]}"/>
</li>
</ol></li>
```

Collections Example: using-collections.xhtml (Continued)

```
<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}"/>
```

Since I use double quotes around the Map key, I use single quotes here.

Collections Example: Input Page Initial Result

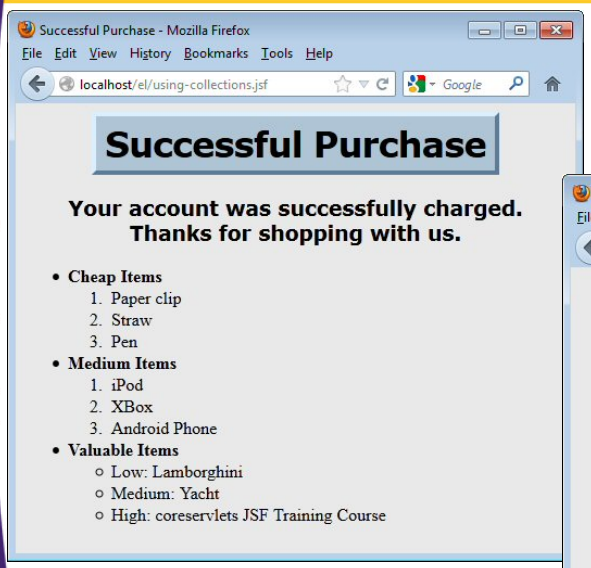


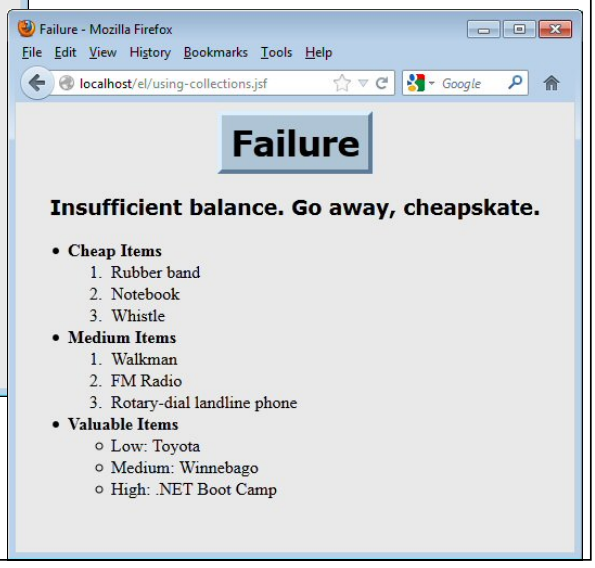
Submitting Properties Example: purchase-success.xhtml

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>
```


Submitting Properties Example: Results





Implicit Objects and Operators

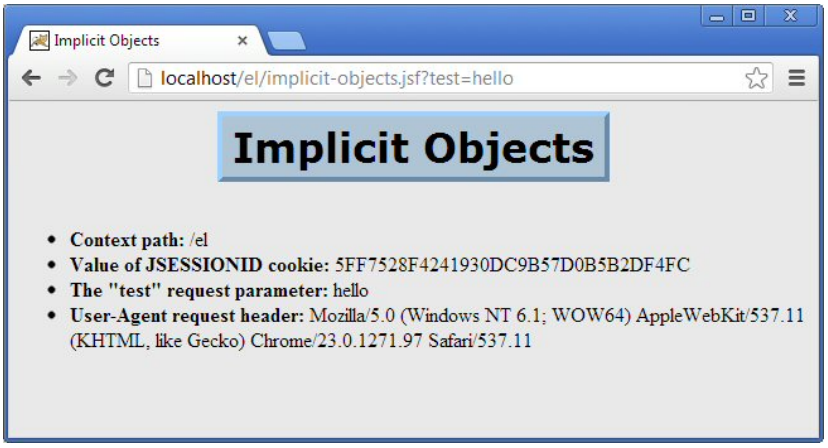
JSF EL Has Almost the Same Predefined Variables as JSP 2

- **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.contextPath}, #{request.queryString}, #{session.id}
      - #{request.contextPath} useful for making relative URLs. See templating section.
  - initParam. Context initialization param.
- **Problem**
  - Using implicit objects works poorly with MVC model. You usually want to use these values in the Java code, not in the facelets pages.

Example: Implicit Objects (Facelets Code)

```
…
<ul>
  <li><b>Context path:</b>
      #{request.contextPath}</li>
  <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

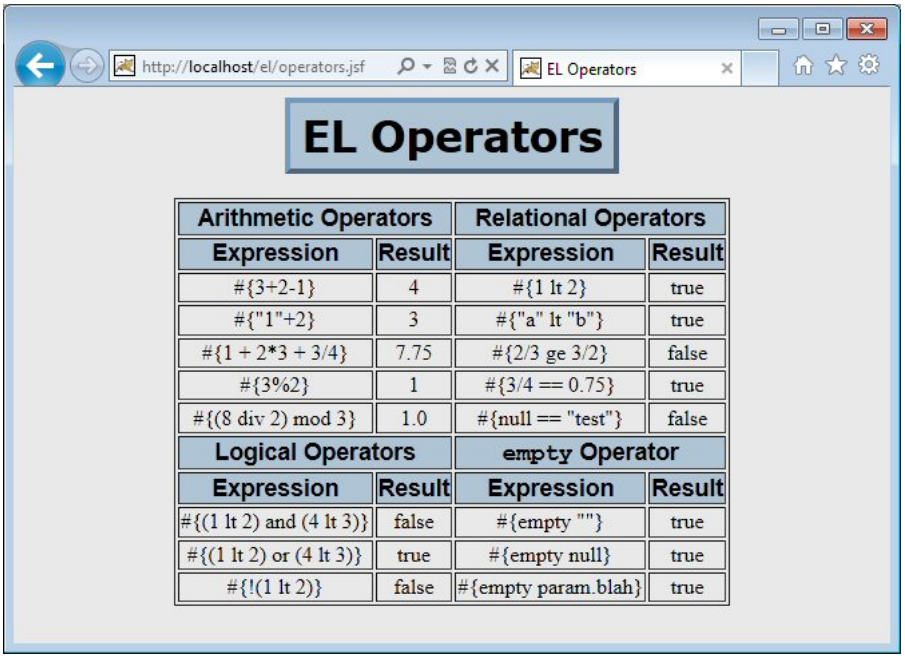
- **Arithmetic**
  - + - * / div % mod
- **Relational**
  - == *or* eq, != *or* ne, < *or* lt, > *or* gt, <= *or* le, >= *or* ge
    - Note: in many contexts in XML, using the operators that contain "<" is illegal. So, you usually use lt instead of <, le instead of <=, etc.
- **Logical**
  - && and || or ! Not
- **Empty**
  - empty
    - True for null, empty string, empty array, empty list, empty map. False otherwise.
- **Note**
  - Use operators sparingly to preserve MVC model

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>
…
```

\#{blah} is taken literally. The backslash prevents EL evaluation.

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 Output

Conditional Text in JSF

- **Alternatives**
  - #{someCondition ? simpleVal1 : simpleVal2}
  - <h:outputText value="#{someValue}"
    rendered="#{someCondition}"/>
    - Or, in general, use h:blah and the “rendered” attribute
  - <ui:fragment rendered="#{someCondition}">
    <someHTML>…</someHTML>
    </ui:fragment>
- **Note**
  - More detailed examples shown in tutorial section on looping in facelets pages

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).

If you want to output HTML, you could use the ternary operator within h:outputText and supply escape="false". But in that case, one of the other two upcoming alternatives is probably simpler.

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"/>

The assumption here is that the getSomeWarning method outputs a string containing HTML tags. If so, the escape="false" is needed to prevent JSF from turning the < into < and so forth.

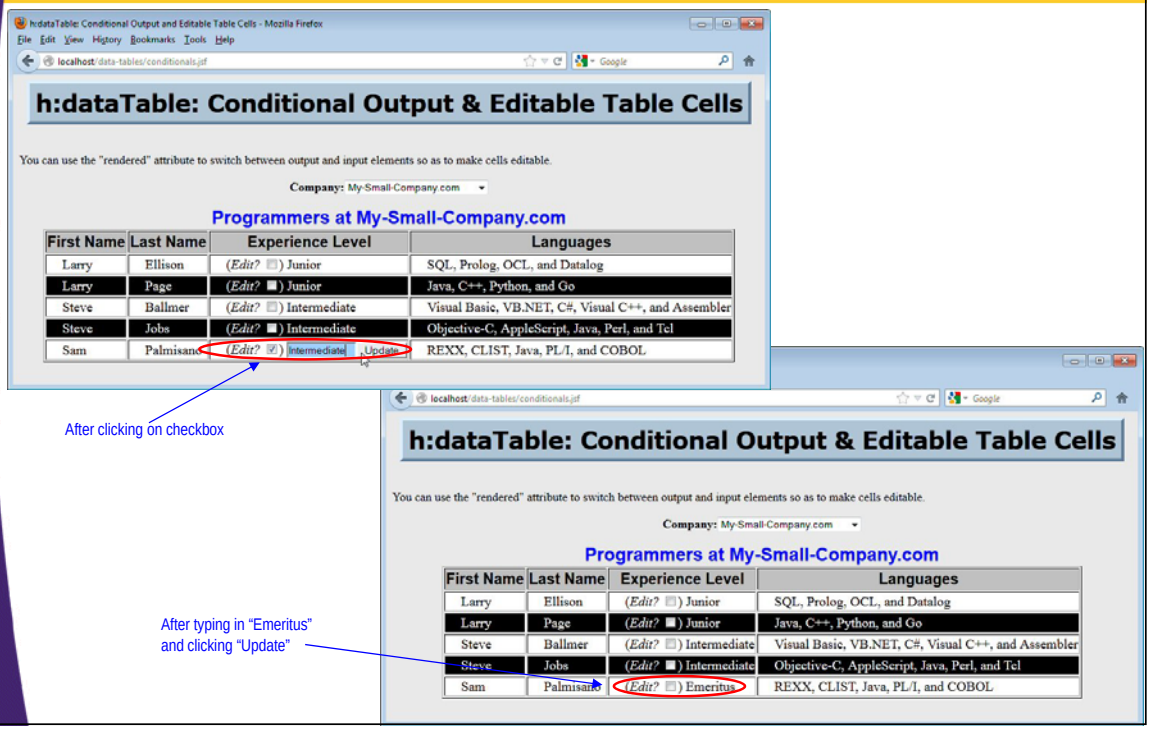
- **When used**
  - When you are outputting simple text (no HTML) or when the HTML comes from a bean.

More on "rendered" Attribute

- **Almost all h:*blah* elements use "rendered"**
  - So, you can insert almost any JSF element conditionally.
- **Example**
  - Insert either textfield followed by button *or* simple value (full example in tutorial section on h:dataTable)

```
<h:inputText value="#{programmer.level}" size="12"
             rendered="#{programmer.levelEditable}"/>
<h:commandButton value="Update"
                 rendered="#{programmer.levelEditable}">
  <f:ajax render="@form" execute="@form"/>
</h:commandButton>
<h:outputText value="#{programmer.level}"
              rendered="#{!programmer.levelEditable}"/>
```

Example: Use of “rendered”



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> Outputs a bold comma after every entry except the last
- **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.
- **Note: define the ui namespace at top**
  - <html … xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

Passing Arguments to Methods

Big Idea

- **EL version 2.2 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 EL 2.2. Not part of JSF 2.0 itself.
    - Server must support servlets 3.0
      - All Java EE 6 servers automatically do
    - So, works in Glassfish 3, JBoss 6, and Tomcat 7. Fails in Tomcat 6, JBoss 5, and other servlet 2.5 engines.

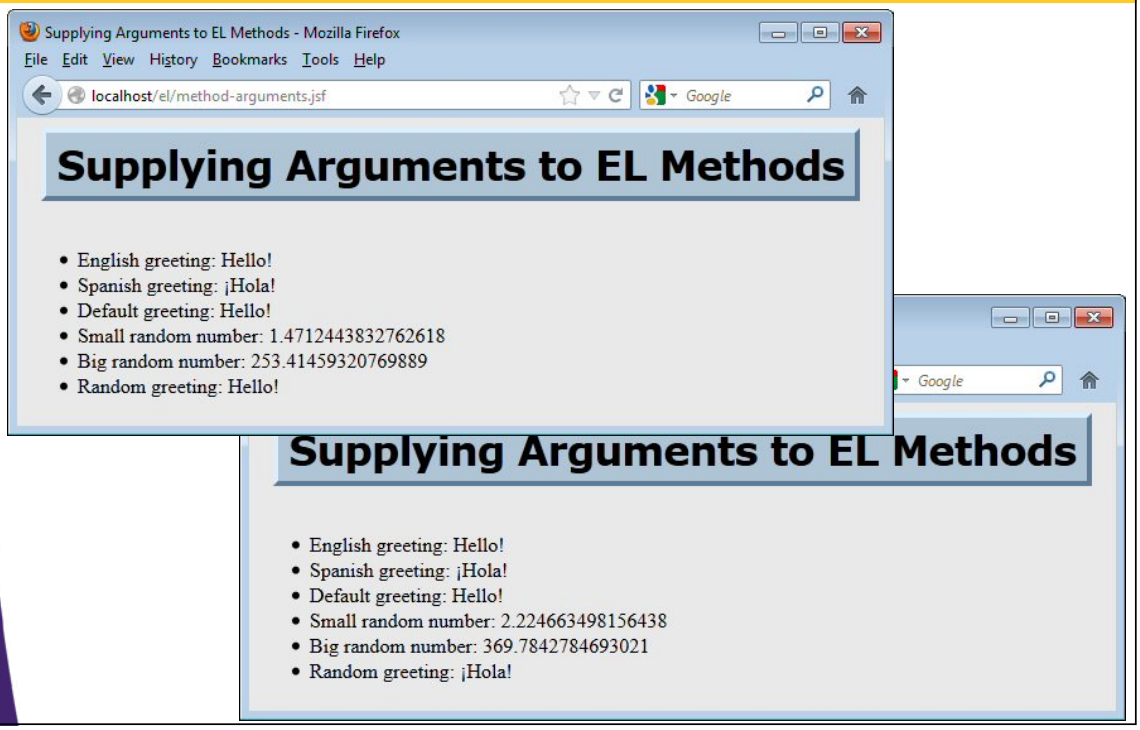
Method Args: Java Code

```
@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



Wrap-Up

Summary

- **Outputting bean properties**
  - #{customer.company.name}
  - <h:outputText value="#{customer.company.name}"/>
    - h:outputText needed only when using rendered, escape, etc.
- **Textfields and other input elements**
  - <h:inputText value="#{customer.firstName}"/>
    - When form displayed, calls getFirstName
    - When form submitted, passes value to setFirstName
- **Collections**
  - #{customer.addresses[0].zip}
    - Call getAddresses, index into array or list, call getZip
    - See also separate tutorial section on looping
- **Operators, conditional evaluation, args**
  - Use for display logic, not for things that could be in Java code

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

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