



ui:repeat and Handling Variable-Length Data

Originals of Slides and Source Code for Examples:
<http://www.coreservlets.com/JSF-Tutorial/jsf2/>

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



For live training on JSF 2.x, please see courses at <http://courses.coreservlets.com/>.



Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
 - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 6 or 7 programming, custom mix of topics
 - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
 - Courses developed and taught by coreservlets.com experts (edited by Marty)
 - Spring, Hibernate/JPA, EJB3, GWT, Hadoop, SOAP-based and RESTful Web Services
- Contact hall@coreservlets.com for details**

Topics in This Section

- **Options for handling variable-length data**
 - Building strings or simple HTML from a bean property
 - Using a builtin component like h:dataTable
 - Making your own composite component
 - Looping with ui:repeat
- **Using ui:repeat**
 - Simple loops
 - Nested loops
 - varStatus
 - Conditional output
 - `{someCondition ? simpleVal1 : simpleVal2}`
 - `<h:outputText rendered="..." .../>`
 - `<ui:fragment rendered="...">...</ui:fragment>`

4

© 2012 Marty Hall



Overview

Customized Java EE Training: <http://courses.coreservlets.com/>


Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Issue

- **Goal**
 - You want results pages to be simple and HTML-oriented
 - Separation of concerns
 - Allows Web page developers to build GUI
- **Problem**
 - What if the action controller method produces data whose length can change? How do you generate output without resorting to JSP scripting and explicit Java looping?
- **Solutions**
 - There are a number of alternatives, *all* of which would work well in some circumstances. The issue is how much control the Web page author needs.
 - We will cover `ui:repeat` in the most detail, but the other alternatives are also reasonable in real life

6

JSF Constructs for Handling Variable-Length Data

- Simplest for Page Author
- 
- Most Control for Page Author
- **Bean**
 - Have bean getter method spit out string or HTML based on a collection
 - **h:dataTable**
 - Use a builtin component that builds a table from a collection. 3rd-party variations such as `t:dataList` (MyFaces/Tomahawk) give even more options.
 - **Your own composite component**
 - Make own component that builds some HTML construct (e.g., `` list) from a collection
 - **ui:repeat**
 - Do explicit looping in results page

7

Competing Concerns

- **Principles**
 - Simplicity is better in the .xhtml pages
 - Layout and formatting decisions should be made by the author of the results page, not by the Java programmer
- **General approach**
 - Use the simplest option that gives the Web page author enough control for the specific situation
- **Notes**
 - We only briefly survey composite components & h:dataTable here. We cover them in detail later in tutorial.
 - Although composite components are simpler than ui:repeat to *use* once they are created, they are more complex to *build* and often use ui:repeat internally. So, we cover them here *after* ui:repeat.

8

When to Use Which: Summary

- **Bean**
 - Write a getter method in the bean that turns collection into plain text or HTML.
 - Programmer knows #{programmer.languageList}.
 - Results in “Programmer knows Java, C++, and Ruby.”
- **When it works well**
 - You output plain text or very simple HTML.
 - You use the same output format several places, with no customization beyond what CSS provides.
- **When it works poorly**
 - The page author needs more control over the output format.

9

When to Use Which: Summary

- **h:dataTable**

- Use the builtin component that turns a collection into an HTML table

- `<h:dataTable var="programmer" value="#{corp.hackers}">
 <h:column>#{programmer.firstName}</h:column>`

- `....
</h:dataTable>`

- **When it works well**

- You want to build an HTML table out of the data
 - Where each entry in data corresponds to a table row

- **When it works poorly**

- You want to build something other than an HTML table
- Different parts of the table come from different sources
 - Although you could make new bean with composite data

10

When to Use Which: Summary

- **Your own composite component**

- Make a new component that turns a collection into HTML (usually something other than a table)

- `<utils:list value="#{programmer.languages}"
 styleClass="some-css-name"/>`

- Result: `<ul class="some-css-name">Java...`

- **When it works well**

- You want to build something other than a table
- You can anticipate the options that page designer needs

- **When it works poorly**

- Data is in a slightly different format than expected
- Page author wants even a small change that was not anticipated by component author

11

When to Use Which: Summary

- **ui:repeat**

- Use facelets looping to build the HTML inside page
 - ``
`<ui:repeat var="language" value="#{person.languages}">`
`#{language}`
`</ui:repeat>`
``

- **When it works well**

- The page designer needs explicit control over the result
- One of the previous options is not sufficient

- **When it works poorly**

- The page becomes so complex that it is hard to maintain by HTML-oriented page author

12

Example Notes

- **Data**

- Normally, the data is produced in the action controller.
 - E.g., you collect a bank customer ID and month in a form, and then the button says
`<h:commandButton ... action="#{user.findChanges}"/>`
where `findChanges` finds the deposits and withdrawals in the month and puts them into an array or List.
- Here, we will hardcode the data for simplicity.

- **Order of topics**

- Your own composite component is listed before `ui:repeat` because it is usually simpler for a page author to *use* a composite component than an explicit loop.
- But, *building* a composite component is more complex and requires `ui:repeat` internally, so is covered after `ui:repeat` in this tutorial.

13



Building Strings or Simple HTML from a Bean

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Big Idea

- **Situation**
 - You have a collection that needs to be displayed as a string or simple HTML element.
- **Approach**
 - Write a getter method in the bean that turns the collection into plain text or HTML.
 - If you output HTML, use a predefined CSS style name.
- **Pros**
 - Very simple for the page author to use.
- **Cons**
 - The page author has very little control over how the output looks (only what CSS allows).
 - Page author can't choose CSS name
 - In long-running apps, changing a Java class might require a restart, whereas changing a .xhtml file does not.

Violating the MVC Model

- **Design strategy behind MVC**
 - Separate the code that creates the data (Java) from the code that presents the data (Facelets)
- **Conclusion?**
 - Java code that builds HTML is always bad
- **Real points**
 - Separation of concerns (loose coupling) is good, and promotes modular apps where various pieces can be changed independently
 - Page developers are good at HTML design, Java programmers are bad at it
 - Page developers need control over their pages

16

I am always nervous about saying this, because I have been preaching the virtues of MVC to servlet and JSP developers for many years now. But nevertheless, using Java to build simple HTML or strings is sometimes OK. But better to err on the side of being too strict about MVC than too loose.

Violating the MVC Model

- **So, HTML in Java is not *always* bad**
 - Done in `h:dataTable`, `h:messages`, `h:outputText` (if it has an id), `h:inputText`, `h:commandButton`, and so on.
 - That's what custom components are all about!
- **Criteria**
 - Reuse.
 - Don't build HTML in your action controller method, but a plain-text String or simple HTML string that is used many places might be worthwhile.
 - Simplicity
 - Usually done for very simple output only
- **Example**
 - `getFullName`: combines `firstName` (e.g., Ed) and `lastName` (e.g., Burns) to produce "Ed Burns"

17

Examples

- **Goal 1**

- Given an array of Strings (or objects with readable toString method), produce a comma-separated English phrase like “Joe, John, and Jane”.

- **Goal 2**

- Same as above, but let the page author customize the fonts and colors. So, produce `Joe, John, and Jane`

18

Example: Programmer Class (Part 1)

```
public class Programmer {
    private String firstName, lastName, level;
    private String[] languages;

    public Programmer(String firstName,
                      String lastName,
                      String level,
                      String... languages) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.level = level;
        this.languages = languages;
    }

    // Getter methods getFirstName, etc. No setters
}
```

19

Example: Programmer Class (Part 2)

```
public String getLanguageList1() {
    StringBuilder langList = new StringBuilder();
    for(int i=0; i<languages.length; i++) {
        if(i < (languages.length-1)) {
            langList.append(languages[i] + ", ");
        } else {
            langList.append("and " + languages[i]);
        }
    }
    return(langList.toString());
}

public String getLanguageList2() {
    String span =
        String.format("<span class='languages'>%s</span>",
            getLanguageList1());
    return(span);
}
```

20

Example: Managed Bean 1

```
package coreservlets;

import javax.faces.bean.*;

@ManagedBean
public class Person1 extends Programmer {
    public Person1() {
        super("Larry", "Ellison", "Junior",
            "SQL", "Prolog", "OCL", "Datalog");
    }
}
```

21

Example: Managed Bean 2

```
package coreservlets;

import javax.faces.bean.*;

@ManagedBean
public class Person2 extends Programmer {
    public Person2() {
        super("Larry", "Page", "Junior",
            "Java", "C++", "Python", "Go");
    }
}
```

22

Example: Facelets Page (Part 1)

```
<!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><title>Building HTML in a Bean</title>
<link href="./css/styles.css"
    rel="stylesheet" type="text/css"/>
<style type="text/css">
    .languages { font-style: italic; }
</style>
</h:head>
<h:body>
...
```

Shown separately from main style sheet for illustration purposes. The page designer needs to know that `getLanguageList2` uses the "languages" CSS name. A composite component (shown later) lets the page author specify the CSS names instead of having them predefined.

23

Example: Facelets Page (Part 2)

```
<h2>First coreservlets.com Developer</h2>
<ul>
  <li>Level: #{person1.level}</li>
  <li>First name: #{person1.firstName}</li>
  <li>Last name: #{person1.lastName}</li>
  <li>Languages: #{person1.languageList1}</li>
</ul>
<h2>Second coreservlets.com Developer</h2>
<p>
Our second #{person2.level}-level developer is
#{person2.firstName} #{person2.lastName}.
He is proficient in
<h:outputText value="#{person2.languageList2}"
  escape="false"/>.
</p>
```

Since `getLanguageList2` outputs HTML, you have to tell JSF not to escape the `<` and `>` (ie, use `<` and `>`). To do this, you use `h:outputText` explicitly and use the "escape" attribute. If "value" is the only attribute used, then `<h:outputText value="#{whatever}"/>` can be replaced by `#{whatever}`.

24

Example: Results



25



Using h:dataTable

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Big Idea

- **Situation**
 - Your data is a List or array, where each entry contains data corresponding to a table row.
- **Approach**
 - Use h:dataTable with one h:column per table column.
 - h:dataTable syntax briefly summarized on next page. Later tutorial section covers h:dataTable in detail.
- **Pros**
 - Flexible and relatively simple way to build a table.
- **Cons**
 - Used only for tables.
 - Data comes from a single source. Source for each column is the same each time around the loop.

Syntax Summary

```
<h:dataTable var="row" Variable name to refer to each entry in collection
    value="#{someBean.someData}" Collection (List, array, or a few other types)
    border="1" It takes effort to use CSS to give table borders, so "border" is supplied as shortcut
    styleClass="css-table-style" CSS style for main table (<table class="...">
    headerClass="css-heading-style" CSS for 1st row, if there is a header defined below
    rowClasses="evenRow,oddRow"> CSS styles for remaining rows, applied alternately

    <h:column>
        <f:facet name="header">Col 1 Title</f:facet> Text for first col in first row
        #{row.someProperty} Value for first col in all subsequent rows
    </h:column>
    <h:column>
        <f:facet name="header">Col 2 Title</f:facet> Text for second col in first row
        #{row.someOtherProperty} Value for second col in all subsequent rows
    </h:column>
    ...
</h:dataTable>
```

28

Example: More Programmers

```
package coreservlets;

import javax.faces.bean.*;

@ManagedBean
public class Person3 extends Programmer {
    public Person3() {
        super("Steve", "Balmer", "Intermediate",
            "Visual Basic", "VB.NET", "C#", "Visual C++",
            "Assembler");
    }
}
```

Person4 and Person5 are similar and not shown.
But all code can be downloaded as an Eclipse
project from tutorial home page (see first slide).

29

Example: Main Managed Bean

```
package coreservlets;

import javax.faces.bean.*;

@ManagedBean(eager=true)
public class Company2 {
    private Programmer[] programmers = {
        new Person1(), new Person2(), new Person3(),
        new Person4(), new Person5()
    };

    public Programmer[] getProgrammers() {
        return(programmers);
    }
}
```

Since we are not reading any form data and this dummy data never changes, we might as well make it application-scoped and pre-instantiated. That is what "eager=true" does.

30

Example: Facelets Page (Part 1)

```
<!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:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">
  <h:head><title>Using h:dataTable</title>
  <link href="./css/styles.css"
        rel="stylesheet" type="text/css"/>
  <link href="./css/table-styles.css"
        rel="stylesheet" type="text/css"/>
</h:head>
<h:body>
...
```

We use f:facet later, so we have to add this namespace

h:dataTable lets page author supply several CSS class names. This stylesheet defines them (contents shown in upcoming slide)

31

Example: Facelets Page (Part 2)

```
<h:dataTable var="programmer"
  value="#{company2.programmers}"
  border="1"
  styleClass="mainTable"
  headerClass="heading"
  rowClasses="evenRow,oddRow">
  <h:column>
    <f:facet name="header">First Name</f:facet>
    #{programmer.firstName}
  </h:column>
  ...
  <h:column>
    <f:facet name="header">Languages</f:facet>
    #{programmer.languageList1}
  </h:column>
</h:dataTable>
```

These styles are defined in table-styles.css (loaded from previous slide, shown on next slide)

32

Example: CSS File (Part 1)

```
.mainTable {
  margin-left: auto;
  margin-right: auto;
}
.heading {
  font-family: sans-serif;
  font-weight: bold;
  font-size: 20px;
  color: black;
  background-color: silver;
  text-align: center;
}
```

33

Example: CSS File (Part 2)

```
.  
  
evenRow {  
    font-family: sans-serif;  
    font-size: 18px;  
    color: black;  
    background-color: white;  
    text-indent: 20px;  
}  
  
.oddRow {  
    font-family: sans-serif;  
    font-size: 18px;  
    color: white;  
    background-color: black;  
    text-indent: 20px;  
}
```

First name given to "rowClasses" in h:dataTable. This will apply to first non-header table row. There is no rule that says there must be two rowClasses entries. Apply the names to the non-header rows until you run out, then repeat. You are also allowed footers and a few other entries that are not covered here but are described in the tutorial section on h:dataTable.

Second name given to "rowClasses" in h:dataTable

34

Example: Results

Using h:dataTable

When your goal is to build an HTML table from the data, h:dataTable often gives you enough flexibility with simpler code than explicitly using ui:repeat.

First Name	Last Name	Experience Level	Languages
Larry	Ellison	Junior	SQL, Prolog, OCL, and Datalog
Larry	Page	Junior	Java, C++, Python, and Go
Steve	Balmer	Intermediate	Visual Basic, VB.NET, C#, Visual C++, and Assembler
Sam	Palmisano	Intermediate	REXX, CLIST, Java, PL/I, and COBOL
Steve	Jobs	Intermediate	Objective-C, AppleScript, Java, Perl, and Tcl

Done

35



Using ui:repeat – Basics

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Big Idea

- **Situation**
 - You have variable-length data, but don't want to output an HTML table.
 - You can't easily build a composite component that gives the page designer enough flexibility.
- **Approach**
 - Use ui:repeat almost exactly as you would use JSTL's c:forEach.
- **Pros**
 - Gives page author explicit control
 - Far simpler and more readable than a JSP scripting loop
- **Cons**
 - Regular HTML has no loops, so page is complex

Syntax Summary

- **Basics**

- `<ui:repeat var="someVar" value="#{someBean.someCollection}">`
 `<someHTML>`
 `#{someVar.someProperty}`
 `</someHTML>`
– `</ui:repeat>`

- **Analogous Java code**

- `for(SomeType someVar: someCollection) {`
 `doSomethingWith(someVar);`
 `}`

- **Warning**

- Cannot use `int[]` or `double[]` or other array of primitives
 - Use `Integer[]` or `Double[]` instead

38

Steps to Using ui:repeat

- **Make sure JSTL 1.2 is available**

- In non-Java-EE-6 server, put JSTL 1.2 JAR files into WEB-INF/lib.

- **Include the facelets namespace**

- `<html ... xmlns:ui="http://java.sun.com/jsf/facelets">`

- **Make collection accessible**

- Make getter method that returns List, array, or ResultSet

- **Use loop in facelets page**

```
<ul>
<ui:repeat var="color" value="#{item.availableColors}">
  <li>#{color}</li>
</ui:repeat>
</ul>
```

39

Why Not JSTL?

- **Question**
 - We know JSTL and `c:forEach`, why learn something new?
- **Answers**
 - `c:forEach` runs when the component tree is being built. `ui:repeat` runs when the tree is being rendered. The latter is when you usually want it to run.
 - `ui:repeat` is virtually identical in syntax and behavior to `c:forEach` anyhow, so if you know `c:forEach`, it is *very* simple to learn `ui:repeat`
 - For the data, just use “value” instead of “items”
- **Caveat**
 - You need `c:forEach` when you want `ui:include` inside a loop, since `ui:include` runs when tree is being built
 - For info on `ui:include`, see tutorial section “Page Templating with Facelets”.

40

Setup

- **JSTL JAR files needed**
 - `ui:repeat` uses JSTL 1.2 internally
 - If you are using a Java EE 6 server, both JSF 2 and JSTL 1.2 are included and no action is needed
 - If you are running in a regular servlet engine like Tomcat, you must add in the JSTL JAR files
 - You already had the JSF JAR files, presumably
- **Downloading**
 - Go to <https://jstl.dev.java.net/download.html>
 - Grab both JAR files
 - Put in WEB-INF/lib



41

Setup (Eclipse Example)

The screenshot shows the Eclipse IDE project structure for a project named 'looping'. The project is expanded to show the following structure:

- looping
 - Deployment Descriptor: looping
 - Java Resources: src
 - JavaScript Resources
 - build
 - WebContent
 - css
 - META-INF
 - resources
 - WEB-INF
 - lib
 - jsf-api.jar
 - jsf-impl.jar
 - jstl-api-1.2.jar
 - jstl-impl-1.2.jar
 - faces-config.xml
 - web.xml
 - bean.xhtml
 - composite-component.xhtml
 - data-table.xhtml
 - index.jsp
 - ui-repeat.xhtml
 - welcome.xhtml

Annotations with arrows pointing to specific files:

- None of these four are needed in Java EE 6 servers (e.g., Glassfish 3) - points to jsf-api.jar, jsf-impl.jar, jstl-api-1.2.jar, and jstl-impl-1.2.jar.
- These two are needed in *all* JSF 2.0 apps that run in non-Java-EE-6 servers. Recall from the intro tutorial that server must support servlets 2.5 or later. - points to jsf-api.jar and jsf-impl.jar.
- These two are needed in JSF 2.0 apps that run in non-Java-EE-6 servers *and* use ui:repeat. - points to jstl-api-1.2.jar and jstl-impl-1.2.jar.

42

Simple Loop: Facelets Code (Top)

```
<!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"
      xmlns:ui="http://java.sun.com/jsf/facelets">
<h:head><title>Looping with ui:repeat</title>
<link href="./css/styles.css"
      rel="stylesheet" type="text/css"/>
<style type="text/css">
  .evenLang { color: blue }
  .oddLang { color: red }
</style>
</h:head>
<h:body>
```

We use ui:repeat and possibly other ui: elements, so we have to add this namespace

These styles are used in a later example (re the varStatus attribute)

43

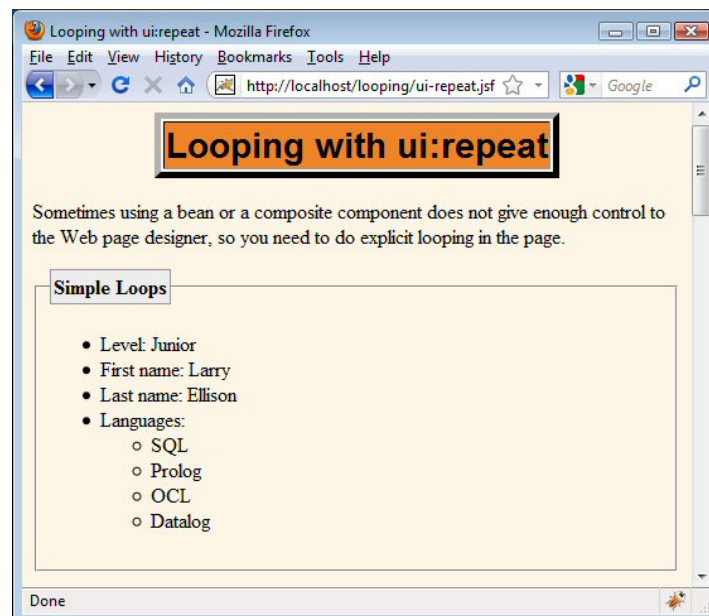
Simple Loop: Facelets Code (ui:repeat Part)

```
<ul>
  <li>Level: #{person1.level}</li>
  <li>First name: #{person1.firstName}</li>
  <li>Last name: #{person1.lastName}</li>
  <li>Languages:
    <ul>
      <ui:repeat var="language" value="#{person1.languages}">
        <li>#{language}</li>
      </ui:repeat>
    </ul>
  </li>
</ul>
```

Code for the person1 managed bean shown earlier.
The getLanguages method returns String[].

44

Simple Loop: Results



45

Nested Loop: Java Code

```
...  
  
@ManagedBean(eager=true)  
public class Company1 {  
    private List<Programmer> programmers;  
  
    public Company1() {  
        programmers = new ArrayList<Programmer>();  
        programmers.add(new Person1());  
        programmers.add(new Person2());  
        programmers.add(new Person3());  
    }  
  
    public List<Programmer> getProgrammers() {  
        return (programmers);  
    }  
}
```

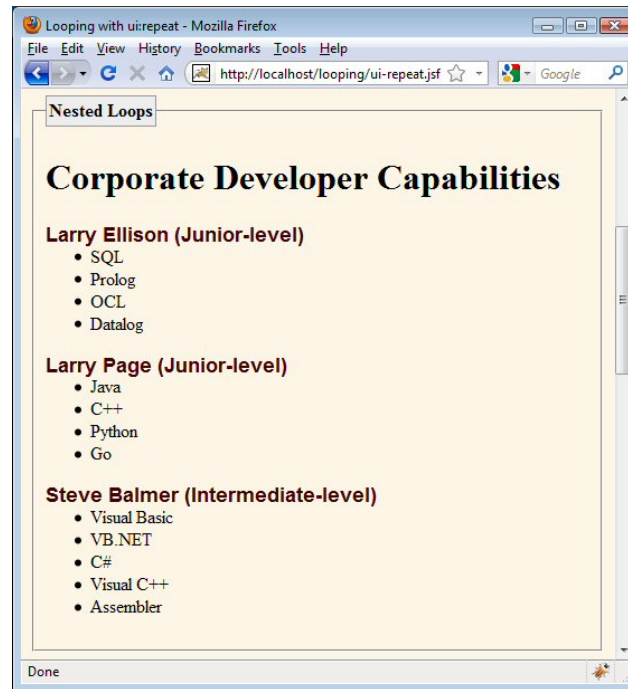
46

Nested Loop: Facelets Code

```
<ui:repeat var="programmer" value="#{company1.programmers}">  
    <h2>#{programmer.firstName} #{programmer.lastName}  
        (#{programmer.level}-level)  
    </h2>  
    <ul style="margin-top: -1em;">  
        <ui:repeat var="language" value="#{programmer.languages}">  
            <li>#{language}</li>  
        </ui:repeat>  
    </ul>  
</ui:repeat>
```

47

Nested Loop: Results



48

© 2012 Marty Hall



Conditional Text Inside Loops

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

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 to JSTL developers**

- `c:if` and `c:choose` don't interact properly with `ui:repeat` because they run when component tree is built, not when it is rendered.
 - So, don't use the JSTL conditional evaluation tags

50

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.

- **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.

51

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.

52

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}"/>
```

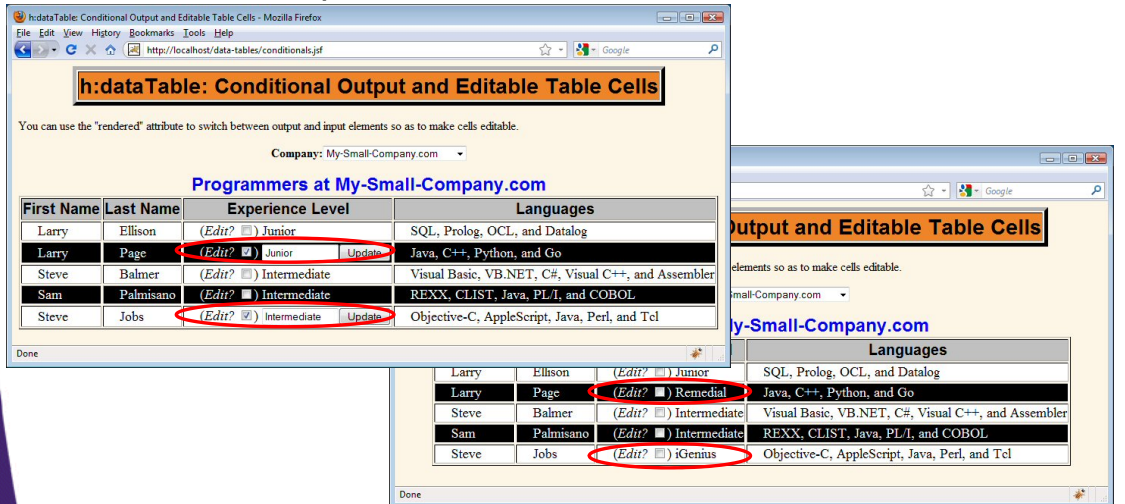
53

Example: Use of “rendered”

- **Idea**

- If checkbox selected, use textfield and “Update” button. Otherwise just show current value.

- Full example in tutorial section on h:dataTable



54

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.

55



Using ui:repeat – Advanced Attributes

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Feature Summary

- **Additional attributes**

```
<ui:repeat var="someVar"  
           value="#{someBean.someCollection}"  
           varStatus="statusVariable"  
           offset="..."  
           size="..."  
           step="...">  
    ...  
</ui:repeat>
```

ui:repeat Attributes

Attribute Name	Description	c:forEach Equivalent
var	String giving the local variable name that will refer to each element in the collection. E.g.: <code><ui:repeat var="name" value="#{party.names}"></code>	var
value	EL expression specifying the collection itself.	items
varStatus	String giving a variable name that will refer to a status object. The status object has the following properties: <ul style="list-style-type: none">• begin, end, step: values of offset, size, and step attributes• index: the current index (int)• first/last: is this the first/last iteration? (boolean)• even/odd: is this even/odd iteration? (boolean) (1st iteration is 0: even) <pre><ui:repeat ... varStatus="status"> Do something with #{status.first} or #{status.even}, etc. </ui:repeat></pre>	varStatus
offset	An int specifying how far into the collection to start. Default is 0. E.g., <code>offset="1"</code> means to skip the first (0 th) element.	begin
size	An int specifying how far down the collection to go. Default is the end of the collection.	end
step	An int specifying how far to jump down the collection after each item. Default is 1.	step

58

Example 1: Overview

- **Goal**
 - From a list of strings, generate “String1, String2, ..., and StringN”.
- **Approach**
 - Track iteration status
 - `<ui:repeat ... varStatus="status">`
 - Output a comma *except* in last iteration
 - `<h:outputText rendered="#{!status.last}" value=","/>`
 - Output “and” *only* in last iteration
 - `<h:outputText rendered="#{status.last}" value=" and "/>`

59

Example 1: Facelets Code

```
<ui:repeat var="programmer" value="#{company1.programmers}">
<h2>#{programmer.firstName} #{programmer.lastName}
    (#{programmer.level}-level)
</h2>
    <ui:repeat var="language" value="#{programmer.languages}"
        varStatus="status">
        <h:outputText value=" and "
            rendered="#{status.last}"/>
        #{language}<h:outputText value=","
            rendered="#{!status.last}"/>
    </ui:repeat>
</ui:repeat>
```

60

Example 1: Results



61

Example 2: Overview

- **Goal**
 - From a list of strings, generate a bulleted () list.
 - Have every other entry in a different style
- **Approach**
 - Track iteration status
 - <ui:repeat ... varStatus="status">
 - Output one type of li in even iterations
 - <ui:fragment rendered="#{status.even}">
 <li class="evenLang">#{language}
 </ui:fragment>
 - Output another type of li in odd iterations
 - <ui:fragment rendered="#{status.odd}">
 <li class="oddLang">#{language}
 </ui:fragment>
 - Use style sheet to map the styles
 - evenLang and oddLang

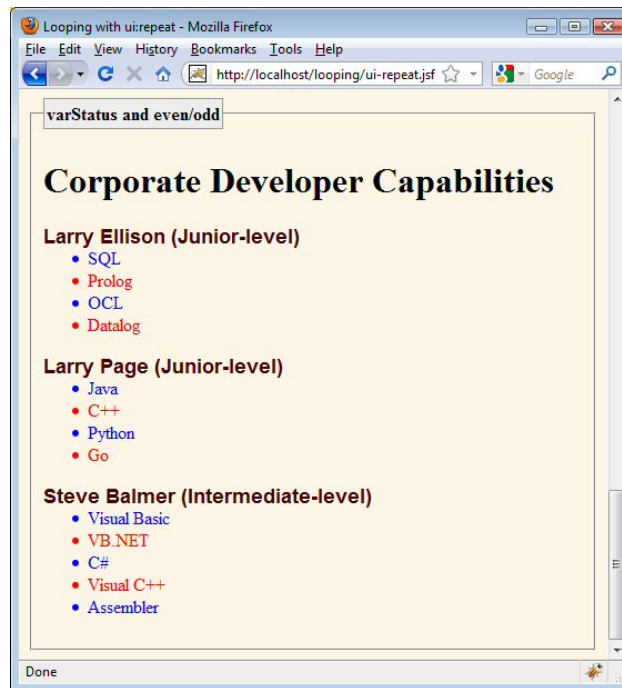
62

Example 2: Facelets Code

```
<ui:repeat var="programmer" value="#{company1.programmers}">
<h2>#{programmer.firstName} #{programmer.lastName}
    (#{programmer.level}-level)
</h2>
<ul style="margin-top: -1em;">
<ui:repeat var="language" value="#{programmer.languages}"
    varStatus="status">
    <ui:fragment rendered="#{status.even}">
        <li class="evenLang">#{language}</li>
    </ui:fragment>
    <ui:fragment rendered="#{!status.even}">
        <li class="oddLang">#{language}</li>
    </ui:fragment>
</ui:repeat>
</ul>
</ui:repeat>
```

63

Example 2: Results



64

Example 2: Alternative

- **Example did this**

```
<ui:fragment rendered="#{status.even}">
  <li class="evenLang">#{language}</li>
</ui:fragment>
<ui:fragment rendered="#{!status.even}">
  <li class="oddLang">#{language}</li>
</ui:fragment>
```

- **It could have done this**

```
<li class="#{status.even ? 'evenLang' : 'oddLang'}">
  #{language}
</li>
```

- **General point**

- If you want to conditionally include large chunks of literal HTML, use ui:fragment, not h:outputText.

65



Making a Composite Component

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Big Idea

- **Situation**
 - You want to output collection in similar way in multiple pages. A no-argument bean getter method does not provide the page author with enough flexibility.
- **Approach**
 - Move the loop into a composite component.
 - Let page author supply collection, CSS names, etc.
- **Pros**
 - Simple for page author to use. Provides moderate flexibility.
- **Cons**
 - Slightly more difficult to create composite component vs. a loop directly in the page. Slightly less flexibility than a loop directly in the page (because the component author must anticipate the options the page author needs).

Overview of Process

- **Define a component**
 - Put `foo.xhtml` in “resources/`bar`” folder
 - Define available attributes inside `composite:interface`
 - Build output inside `composite:implementation`
 - This part probably uses `ui:repeat`
- **Use component namespace in facelets page**
 - `xmlns:bar="http://java.sun.com/jsf/composite/bar"`
- **Use the component**
 - `<bar:foo ... />`
- **Note**
 - Composite components covered in detail in later tutorial section.

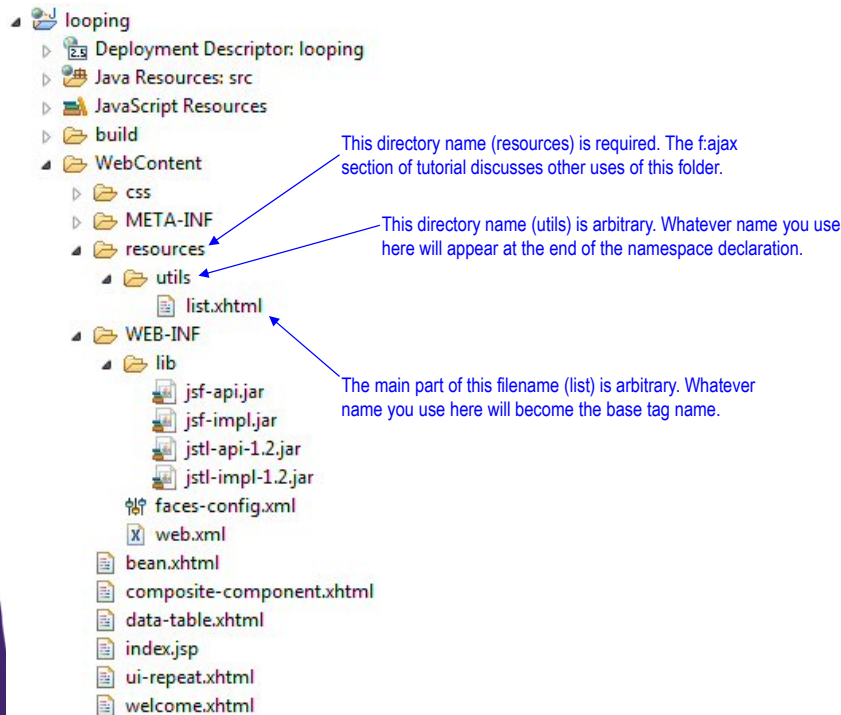
68

Example: Overview

- **Goal**
 - You want to turn a collection into a bulleted (``) list in several places. You want to let the page author specify the CSS style for the list.
- **Approach**
 - Define a composite component: `resources/utills/list.xhtml`
 - In `composite:interface`, define “`styleClass`” and “`value`” attributes
 - In `composite:implementation`, use `ui:repeat` to build ul list
 - Use namespace
 - `xmlns:utills="http://java.sun.com/jsf/composite/utills"`
 - Use component
 - `<utills:list styleClass="someStyle" value="#{someData}"/>`
 - Define `someStyle` in CSS file

69

Example: Eclipse Layout



70

Example: Component Structure (WebContent/resources/utils/list.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"
  xmlns:composite="http://java.sun.com/jsf/composite"
  xmlns:ui="http://java.sun.com/jsf/facelets">
  <head><title>(For validation only)</title></head>
  <body>
    <composite:interface>...</composite:interface>
    <composite:implementation>...</composite:implementation>
  </body></html>
```

Needed in all composite components.

Needed in this case because this composite component uses ui:repeat.

These two parts are shown on next page. Only the part inside composite:implementation becomes part of the output.

71

Example: Component Body (WebContent/resources/utils/list.xhtml)

```
...
<composite:interface>
  <composite:attribute name="value"/>
  <composite:attribute name="styleClass"/>
</composite:interface>

<composite:implementation>
  <ul class="#{cc.attrs.styleClass}">
    <ui:repeat var="listItem" value="#{cc.attrs.value}">
      <li>#{listItem}</li>
    </ui:repeat>
  </ul>
</composite:implementation>
...
```

cc is predefined variable with predefined attrs property. So, cc.attrs.styleClass means "the value passed in by page author for the styleClass attribute".

72

Example: Facelets Page (Top)

```
<!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"
  xmlns:utils="http://java.sun.com/jsf/composite/utils">
<h:head><title>Using a Composite Component</title>
<link href="./css/styles.css"
  rel="stylesheet" type="text/css"/>
<style type="text/css">
  .langs1 { font-style: italic; }
  .langs2 { font-weight: bold; color: blue; margin-top: -1em; }
</style>
```

CSS names used in utils:list on next pages. Put here instead of in separate stylesheet file for illustration.

Matches the folder name (under "resources") of the component.

Technically the namespace at the left can be something different, but it is common to use same name both places. Whatever you use for namespace you use in tag on next page.

73

Example: Facelets Page (Body)

```
<h2>First coreservlets.com Developer</h2>
<ul>
  <li>Level: Junior</li>
  <li>First name: #{person1.firstName}</li>
  <li>Last name: #{person1.lastName}</li>
  <li>Languages: <utils:list value="#{person1.languages}"
                    styleClass="langs1"/></li>
</ul>

<h2>Second coreservlets.com Developer</h2>
<p>
Our second junior-level developer is
#{person2.firstName} #{person2.lastName}.
He is proficient in:
<utils:list value="#{person2.languages}"
            styleClass="langs2"/>
</p>
```

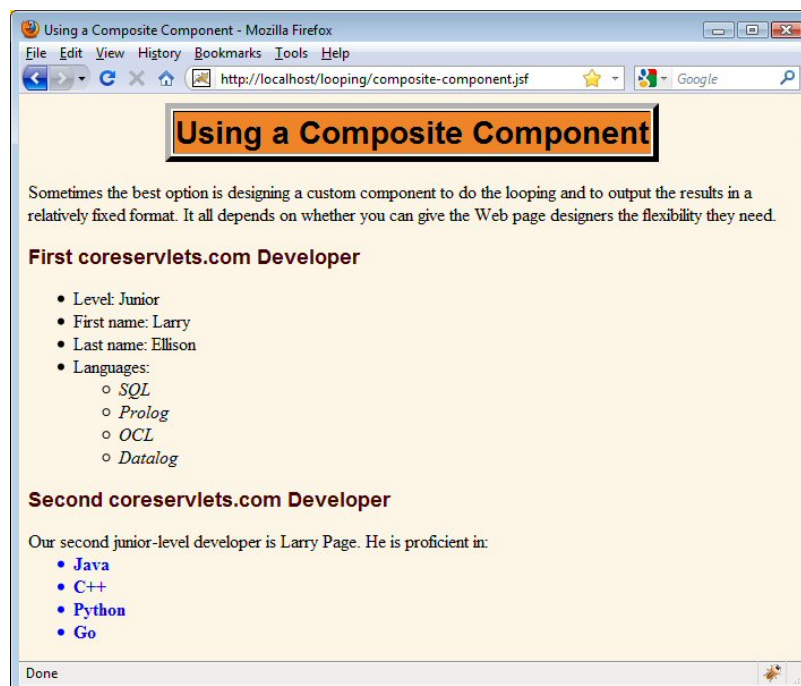
"utils" matches the namespace from previous page (which is often the folder name within "resources")

"list" matches the base filename of the component (i.e., list.xhtml)

The two attributes (value and styleClass) were defined in composite:interface in the component file.

74

Example: Results



75



Wrap-Up

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Summary

- **ui:repeat basics**

```
<ul>  
  <ui:repeat var="color" value="#{item.availableColors}">  
    <li>#{color}</li>  
  </ui:repeat>  
</ul>
```
- **Other ui:repeat capabilities**
 - varStatus attribute
 - Especially first, last, even, and odd boolean properties
 - Conditional output
 - Use “rendered” attribute of h:outputText or ui:fragment
- **Consider alternatives to ui:repeat**
 - Bean getter method that builds result
 - h:dataTable
 - Composite component
 - Especially one that uses ui:repeat internally



Questions?

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.