JSF Data Table

Agenda

- Motivation
- Basic syntax
- Defining table headings
- Formatting tables with style sheets
- Displaying database tables

Dealing with Variable Length Data

- **Issue**
  - What if the business/data-access logic creates something with an indeterminate number of elements? How do you output it in the final JSP page without breaking the MVC model?
- **Alternatives**
  - Non-looping
    - <h:outputText value="#{bankCustomer.depositTable}"/>
    - <mytags:showDepositTable custID="..." month="..." styleClasses="..."/>
  - Looping

    ```
     <% for(...) { ... %>
    HTML Code
    <% } %>
    ```
  - JSTL

Building HTML Tables

- **Assume you need HTML exposed in JSP page, so you cannot use the non-looping alternatives**
  - Using JSP scripting elements to loop is unwieldy
  - Using JSTL to loop is an option, but is still complex
- **JSF provides h:dataTable**
  - You give *one* row definition, and JSF repeats it for you

```
<h:dataTable value="#{someBean.someCollection}"
  var="rowVar" border="1">
<h:column>
  <h:outputText value="#{rowVar.col1Data}"/>
</h:column>
<h:column>
  <h:outputText value="#{rowVar.col2Data}"/>
</h:column>
...
</h:dataTable>
```

Details: h:dataTable

- **value: a collection of data (list of beans, usually). Legal collection types:**
  - Array
  - List (e.g., ArrayList, LinkedList)
  - ResultSet (must be scroll-insensitive)
  - Result (wrapped ResultSet from JSTL)
  - DataModel (in javax.faces.model)
- **var: bound to each collection entry in turn**
  - This entry should be something the JSF EL can output
    - Bean, array, List, Map
- **Other attributes**
  - Standard TABLE attributes
    - border, bgcolor, width, cellpadding, cellspacing, frame, ...
  - Style sheet designators
    - rowClasses, headerClass, footerClass

Details: h:column

- Usually encloses h:outputText elements which reference the variable from the enclosing h:dataTable

```
<h:column>
  <h:outputText value="#{rowVar.colData}"/>
</h:column>
```

- Can enclose other h:Xxxx elements
  - E.g. h:inputText, or any other
- Regular HTML content must be enclosed in f:verbatim

```
<h:column>
  <f:verbatim>First Name: </f:verbatim>
<h:outputText value="#{rowVar.firstName}"/>
</h:column>
```

- Table headings and footers specified with f:facet
  - See later example

Example: Printing Table of Sales Based on Array

- SalesBean class represents sales of apples and oranges (in dollars)
- Array of SalesBean objects represents quarterly sales in year
- All values to be presented in HTML table

SalesBean: Basic Code

```
public class SalesBean {
  private double apples = 0.0, oranges = 0.0;
  public SalesBean() {}
  public SalesBean(double apples, double oranges) {
    setApples(apples);
    setOranges(oranges);
  }
  public double getApples() { return(apples); }
  public void setApples(double apples) {
    this.apples = apples;
  }
  public double getOranges() { return(oranges); }
  public void setOranges(double oranges) {
    this.oranges = oranges;
  }
  public SalesBean[] getYearlySales() {
    SalesBean[] yearlySales =
      { new SalesBean(100.22, 200.32),
        new SalesBean(300.44, 400.55),
        new SalesBean(500.66, 600.77),
        new SalesBean(700.88, 800.99) };
    return(yearlySales);
  }
}
```

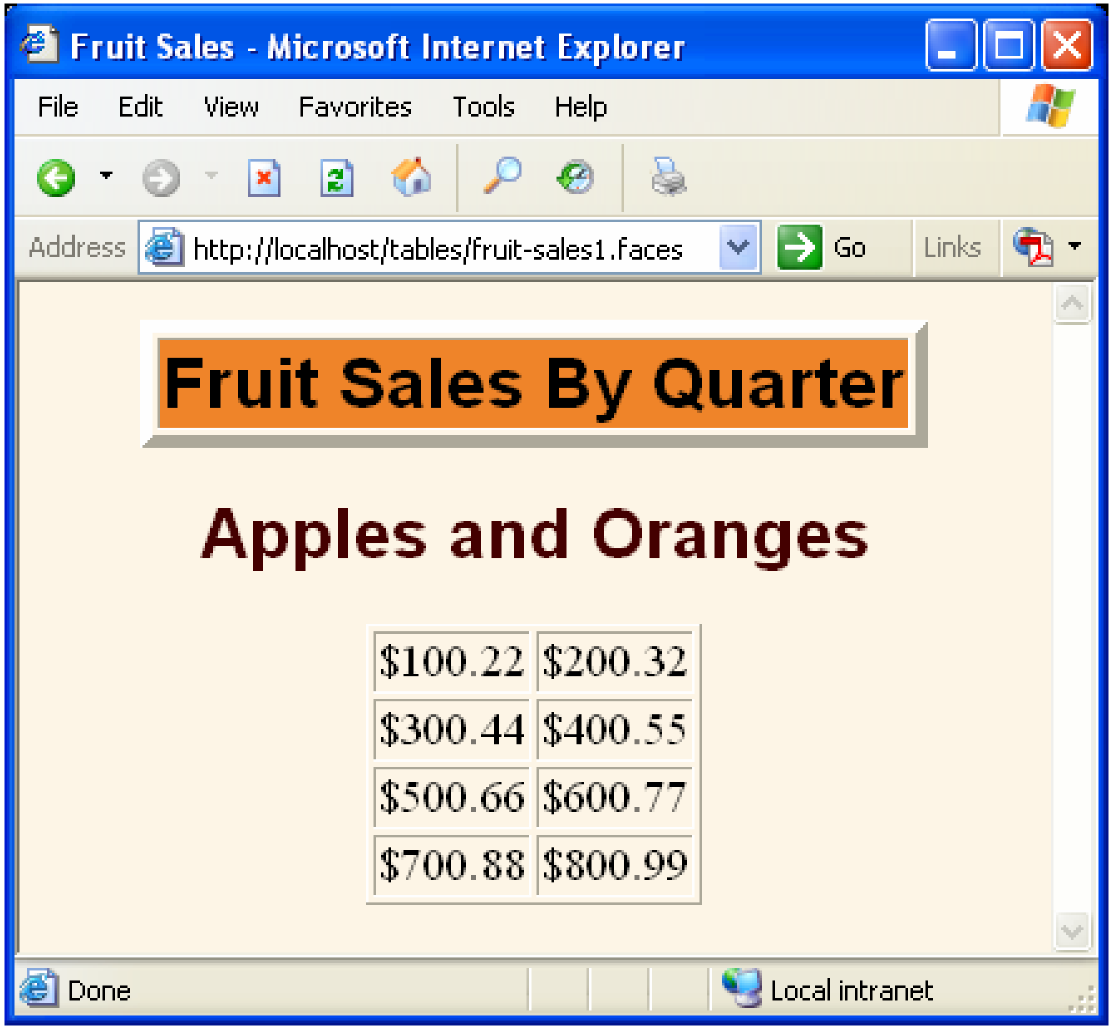
Faces-config.xml

```
……..
<faces-config>
        <managed-bean>
        <managed-bean-name>salesBean</managed-bean-name>
        <managed-bean-class>SalesBean</managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
        </managed-bean>
...
</faces-config>
```

fruit-sales1.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
...
<H2>Apples and Oranges</H2>
<h:dataTable value="#{salesBean.yearlySales}“ var="quarterlySales"
                                                    border="1">
<h:column>
    <f:verbatim>$</f:verbatim>
    <h:outputText value="#{quarterlySales.apples}"/>
</h:column>
<h:column>
        <f:verbatim>$</f:verbatim>
        <h:outputText value="#{quarterlySales.oranges}"/>
</h:column>
</h:dataTable>
...
</f:view>
```

Fruite-sales1.faces


Fruit Sales - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
http://localhost/tables/fruit-sales1.faces
Go
Links
Done
Local intranet

Fruit Sales By Quarter

Apples and Oranges

$100.22	$200.32
$300.44	$400.55
$500.66	$600.77
$700.88	$800.99

Defining Table Headings

Headers and Footers

- **Headers**
  - Use f:facet with name="header"
  - Value can be f:verbatim or h:outputText
  - Still need h:outputText for non-heading value
- **Footers**
  - Use f:facet with name="footer"
  - Value can be f:verbatim or h:outputText
- **Example Code**

```
<h:column>
  <f:facet name="header">
  <f:verbatim>...</f:verbatim>
</f:facet>
  <h:outputText value="#{rowVar.colVal}"/>
</h:column>
```

Formatting Tables with Style Sheets

Options

- **Use explicit formatting in JSP**
  - Requires f:verbatim before and after each entry
  - Long and clumsy
- **Embed formatting in bean**
  - Not accessible to Web developer
  - Risks inconsistencies with style sheet
  - Beans are for model, not view
- **Use rowClasses, headerClass, footerClass**
  - **rowClasses**: comma-separated list of CSS styles. Applied to each row until list ends, then repeats
  - **headerClass**: CSS style for heading
  - **footerClass**: CSS style for footer

Database-Driven Tables

Options for Displaying Database Results

- **Extract data from result set and place in array or List**
  - Tedious
  - Requires a bean to represent a row of data
- **Use ResultSet as arg to value of h:dataTable**
  - Should work, according to spec
    - In practice, you must use scroll-insensitive or it crashes
  - ResultSet is connected, so a big pain to go back and close connections (or return them to pool) later
- **Use JSTL ResultSupport class to turn ResultSet into Result**
  - Only one extra line of code
  - Robust and reliable
  - Result is disconnected, so you can close connection or return it to pool before using the Result

ResultSet vs Result

- java.sql.ResultSet
  - low level
- javax.servlet.jsp.jstl.sql.ResultSet
  - a bean that wraps a result set and implements programming control
  - easier to use than ResultSet

Example:
Displaying Data using <h:dataTable> from ResultSet

The Model class

```
public class Model {
ResultSet rs = null;
public Model() throws Exception {
getDataFromDb(); }
public void getDataFromDb() throws Exception {
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection(
"jdbc:oracle:thin:@192.168.4.242:1521:orcl", "scott",
"tiger");
Statement st =
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
rs = st.executeQuery("select * from emp");
if (rs != null) {
System.out.println("The data is loaded");
}}
public ResultSet getRs() {
return rs;}
public void setRs(ResultSet rs) {
this.rs = rs;}}
```

faces-config.xml

```
<faces-config>
<managed-bean>
<managed-bean-name>data</managed-bean-name>
<managed-bean-class>
      com.jp.jsf.Model
</managed-bean-class>
<managed-bean-scope>request</managed-bean-scope>
</managed-bean>
</faces-config>
```

Showdata.jsp (1/2)

```
<%@ page language="java" pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
		<head>
			<title>My JSF 'showdata.jsp' starting page</title>
<style type="text/css">
.header{
background-color:brown;
color: white;
}
</style>
```

Showdata.jsp (2/3)

```
</head>
        <body>
        <f:view>
        <h:dataTable value="#{data.rs}" var="row" cellpadding="2" border="1">
                <h:column>
                        <f:facet name="header" >
                        <h:outputText value="ID" styleClass="header" />
                        </f:facet>
                        <h:outputText value="#{row.EMPNO}" />
                </h:column>
                <h:column>
                        <f:facet name="header" >
                        <h:outputText value="NAME" styleClass="header"/>
                        </f:facet>
                        <h:outputText value="#{row.ENAME}" />
                </h:column>
```

Showdata.jsp (3/3)

```
                <h:column>
                <f:facet name="header">
                <h:outputText value="JOB" styleClass="header"/>
                </f:facet>
                <h:outputText value="#{row.JOB}" />
                </h:column>
                <h:column>
                <f:facet name="header">
                <h:outputText value="Salary" styleClass="header" />
                </f:facet>
                <h:outputText value="#{row.SAL}" />
                </h:column>
                </h:dataTable>
                </f:view>
        </body>
</html>
```

DataModel wrapper

- All data sources for UIData components have a DataModel wrapper
- Unless you explicitly construct a DataModel wrapper, the JavaServer Faces implementation will create one around data of any of the other acceptable types

Types of Data Models

- ArrayDataModel
- ListDataModel
- ResultDataModel
- ResultSetDataModel
- ScalarDataModel

Adding Editable Components (1/2)

- Basic idea behind making a component like **outputText** to editable is making the **outputText** field to disappear and a field type **inputText** appear in the former’s place.
- In this technique we make use of the attribute “rendered”
- The value of **rendered** attribute is set to **“true”** or **“false”** as required through a boolean flag using an “event”.

Adding Editable Components (1/2)

```
<h:dataTable value="#{tableData.names}" var="name">
          <h:column>
            <f:facet name="header">
              <h:outputText value="#{msgs.editColumn}"
                style="font-weight: bold"/>
            </f:facet>
            <h:selectBooleanCheckbox value="#{name.editable}"
              onclick="submit()"/>
          </h:column>
          <h:column>
            <f:facet name="header">
              <h:outputText value="#{msgs.lastnameColumn}"
                style="font-weight: bold"/>
            </f:facet>
            <h:inputText value="#{name.last}" rendered="#{name.editable}"
              size="10"/>
            <h:outputText value="#{name.last}" rendered="#{not name.editable}"/>
          </h:column>
……………………….
</h:dataTable>
```

Sorting and Scrolling

Sorting and Scrolling

- Will be handled in the class