Anno Accademico 2007-2008

Corso di Tecnologie Web
Web Application: Java Server Faces
DataTable

http://www-lia.deis.unibo.it/Courses/TecnologieWeb0708/

JSF: DataTable

> Le tabelle HTML sono utilizzate per rappresentare contenuti in forma matriciale bidimensionale

> JSF supporta il rendering HTML di tali elementi mediante il tag **`h:dataTable`**

> L'attributo **`value`** del tag **`h:dataTable`** rappresenta la struttura dati su cui iterare per ottenere le righe della tabella (**`h:dataTable`** è *row oriented*); tale valore si esprime, come per gli altri tag, utilizzando JSF EL. Il tipo della property agganciata dall'espressione EL dev'essere:

- un array
- un'istanza di **`java.util.List`**
- un'istanza di **`java.sql.ResultSet`**
- un'istanza di **`javax.servlet.jsp.jstl.sql.Result`**
- un'istanza di **`javax.faces.model.DataModel`**

JSF: DataTable – un semplice esempio

```
 1. <html>
 2.    <%@ taglib uri="http://java.sun.com/jsf/core"  prefix="f" %>
 3.    <%@ taglib uri="http://java.sun.com/jsf/html"  prefix="h" %>
 4.    <f:view>
 5.       <head>
 6.          <f:loadBundle basename="messages" var="msgs"/>
 7.          <title>
 8.             <h:outputText value="#{msgs.windowTitle}"/>
 9.          </title>
10.       </head>
11.       <body>
12.          <h:outputText value="#{msgs.pageTitle}"/>
13.          <p>
14.          <h:form>
15.             <h:dataTable value="#{tableData.names}"
16.                          var="name">
17.                <h:column>
18.                   <h:outputText value="#{name.last}"/>
19.                   <f:verbatim>,</f:verbatim>
20.                </h:column>
21.
22.                <h:column>
23.                   <h:outputText value="#{name.first}"/>
24.                </h:column>
25.             </h:dataTable>
26.          </h:form>
27.       </body>
28.    </f:view>
29. </html>
```

JSF: DataTable – un semplice esempio

```
package it.unibo.deis;

public class Name {
   private String first;
   private String last;

   public Name(String first, String last) {
      this.first = first;
      this.last = last;
   }

   public void setFirst(String newValue) { first = newValue; }
   public String getFirst() { return first; }

   public void setLast(String newValue) { last = newValue; }
   public String getLast() { return last; }
}
```

```
package it.unibo.deis;

public class TableData {
   private static final Name[] names = new Name[] {
       new Name("William", "Dupont"),
       new Name("Anna", "Keeney"),
       new Name("Mariko", "Randor"),
       new Name("John", "Wilson")
   };
   public Name[] getNames() { return names;}
}
```

JSF: DataTable – un semplice esempio

> Il corpo di **`h:dataTable`** contiene solo tag **`h:column`**; **`h:dataTable`** ignora tutti gli altri componenti. Una colonna può contenere un numero illimitato di tag oltre agli opzionali *header* e *footer*

> **`h:dataTable`** combina il componente **`UIData`** con il renderer **`Table`**. Questa combinazione genera una tabella HTML che garantisce il supporto a tutte le impostazioni stilistiche HTML, nonché l'accesso a DB, custom data model, ecc.

JSF: header e footer

> Header e footer vengono renderizzati utilizzando il tag core **`f:facet`**

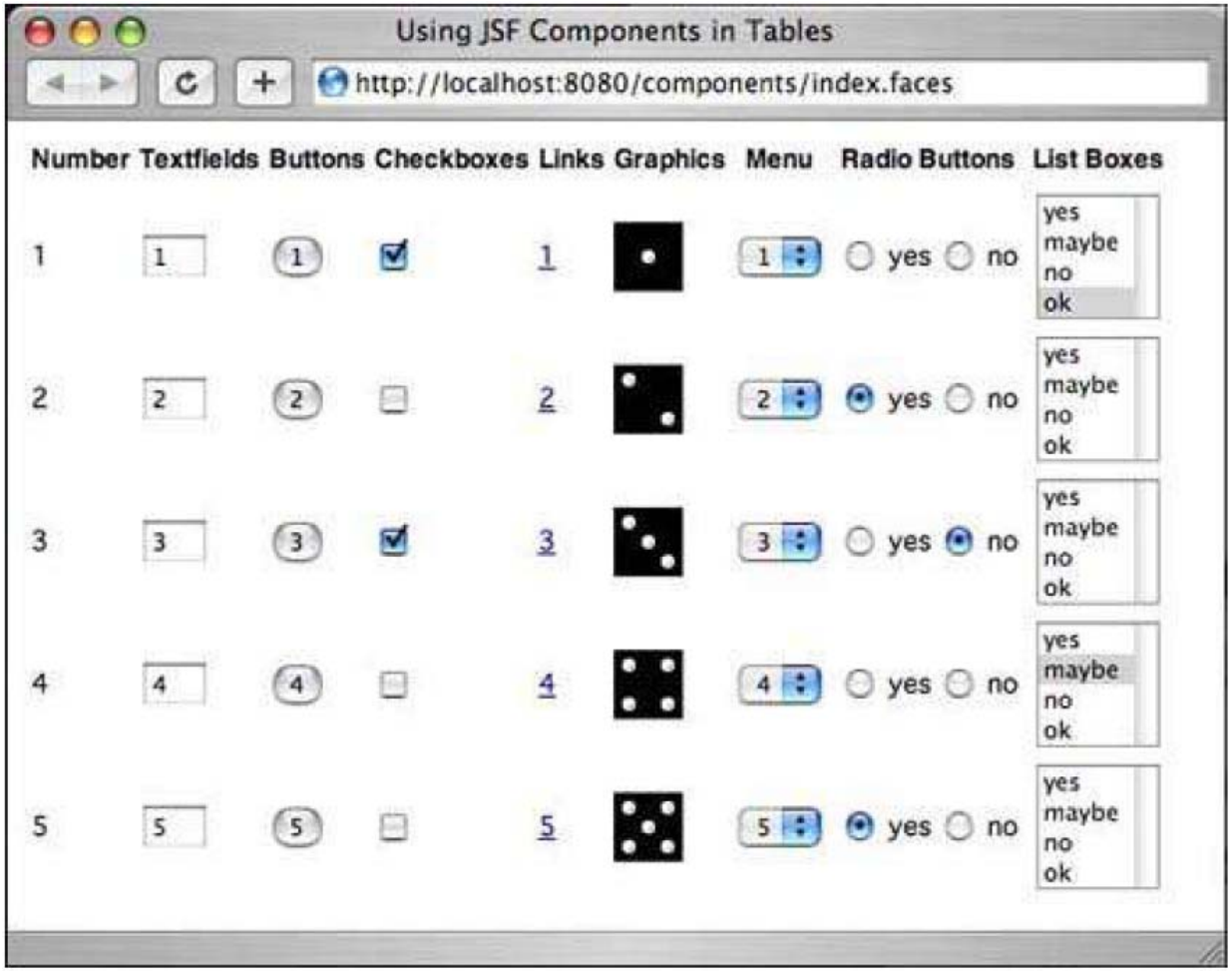
```
<h:dataTable>
    ...
    <h:column>
        <f:facet name="header">
            <%-- header components go here --%>
        </f:facet>

        <%-- column components go here --%>

        <f:facet name="footer">
            <%-- footer components go here --%>
        </f:facet>
    </h:column>
    ...
</h:dataTable>
```

JSF: header e footer

> Vediamo ora un esempio di utilizzo di header e footer in una tabella che contiene anche componenti di input nelle celle



JSF: header e footer - esempio

```
 1. <html>
 2.    <%@ taglib uri="http://java.sun.com/jsf/core"  prefix="f" %>
 3.    <%@ taglib uri="http://java.sun.com/jsf/html"  prefix="h" %>
 4.    <f:view>
 5.       <head>
 6.          <link href="styles.css" rel="stylesheet" type="text/css"/>
 7.          <title>
 8.             <f:loadBundle basename="messages" var="msgs"/>
 9.             <h:outputText value="#{msgs.windowTitle}"/>
10.          </title>
11.       </head>
12.       <body>
13.          <h:form>
14.             <h:dataTable value="#{numberList}" var="number">
15.                <h:column>
16.                   <f:facet name="header">
17.                      <h:outputText value="#{msgs.numberHeader}"/>
18.                   </f:facet>
19.                   <h:outputText value="#{number}"/>
20.                </h:column>
21.                <h:column>
22.                   <f:facet name="header">
23.                      <h:outputText value="#{msgs.textfieldHeader}"/>
24.                   </f:facet>
25.                   <h:inputText value="#{number}" size="3"/>
26.                </h:column>
27.                <h:column>
28.                   <f:facet name="header">
29.                      <h:outputText value="#{msgs.buttonHeader}"/>
30.                   </f:facet>
31.                   <h:commandButton value="#{number}"/>
32.                </h:column>
```

JSF: header e footer - esempio

```
33.                 <h:column>
34.                    <f:facet name="header">
35.                       <h:outputText value="#{msgs.checkboxHeader}"/>
36.                    </f:facet>
37.                    <h:selectBooleanCheckbox value="false"/>
38.                 </h:column>
39.                 <h:column>
40.                    <f:facet name="header">
41.                       <h:outputText value="#{msgs.linkHeader}"/>
42.                    </f:facet>
43.                    <h:commandLink>
44.                       <h:outputText value="#{number}"/>
45.                    </h:commandLink>
46.                 </h:column>
47.                 <h:column>
48.                    <f:facet name="header">
49.                       <h:outputText value="#{msgs.graphicHeader}"/>
50.                    </f:facet>
51.                    <h:graphicImage value="images/dice#{number}.gif"
52.                       style="border: 0px"/>
53.                 </h:column>
54.                 <h:column>
55.                    <f:facet name="header">
56.                       <h:outputText value="#{msgs.menuHeader}"/>
57.                    </f:facet>
58.                    <h:selectOneMenu>
59.                       <f:selectItem itemLabel="#{number}" itemValue="#{number}"/>
60.                    </h:selectOneMenu>
61.                 </h:column>
```

JSF: header e footer - esempio

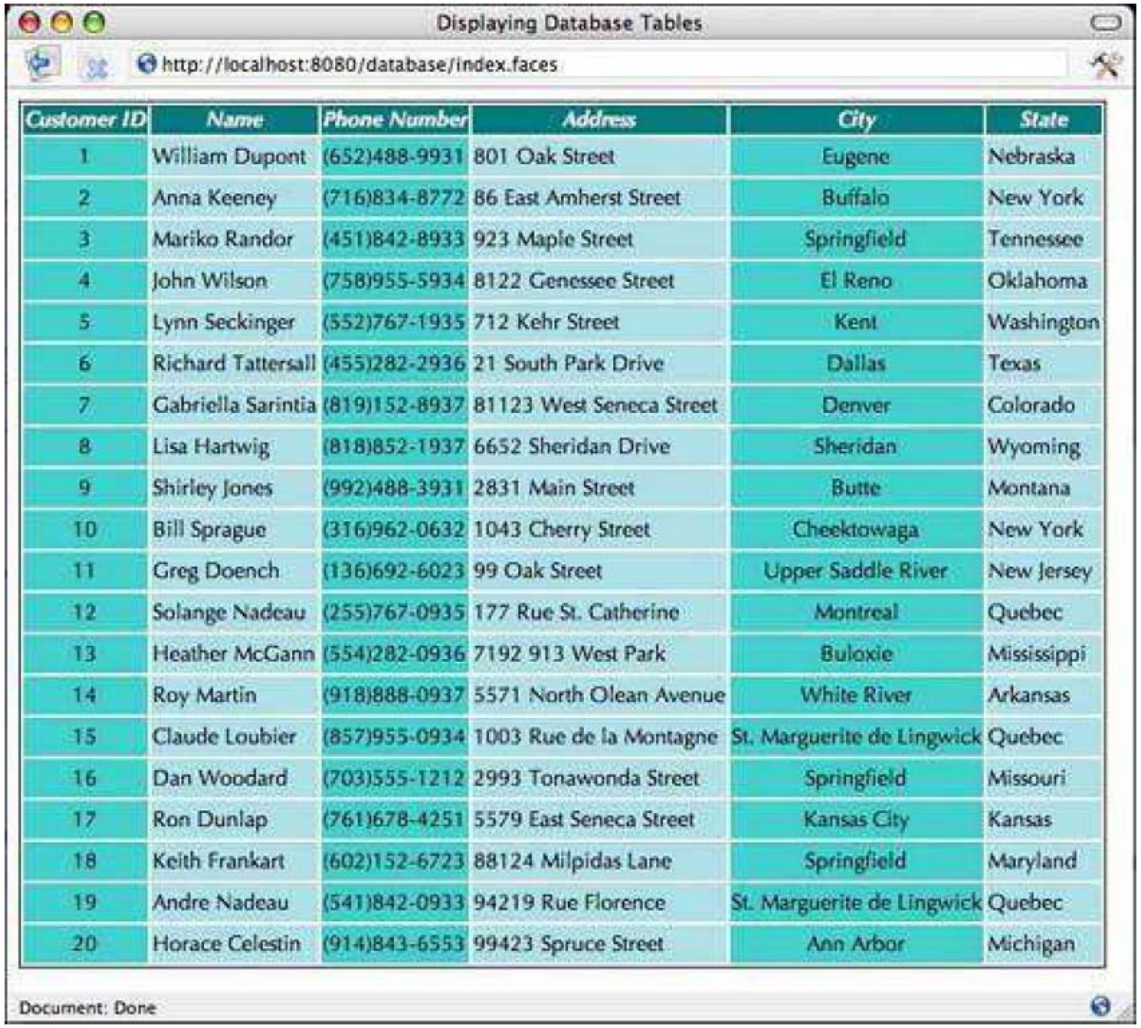
```
62.                 <h:column>
63.                    <f:facet name="header">
64.                       <h:outputText value="#{msgs.radioHeader}"/>
65.                    </f:facet>
66.                    <h:selectOneRadio layout="LINE_DIRECTION" value="nextMonth">
67.                       <f:selectItem itemValue="yes" itemLabel="yes"/>
68.                       <f:selectItem itemValue="no" itemLabel="no" />
69.                    </h:selectOneRadio>
70.                 </h:column>
71.                 <h:column>
72.                    <f:facet name="header">
73.                       <h:outputText value="#{msgs.listboxHeader}"/>
74.                    </f:facet>
75.                    <h:selectOneListbox size="3">
76.                       <f:selectItem itemValue="yes" itemLabel="yes"/>
77.                       <f:selectItem itemValue="maybe" itemLabel="maybe"/>
78.                       <f:selectItem itemValue="no" itemLabel="no" />
79.                       <f:selectItem itemValue="ok" itemLabel="ok" />
80.                    </h:selectOneListbox>
81.                 </h:column>
82.              </h:dataTable>
83.           </h:form>
84.        </body>
85.     </f:view>
86. </html>
```

JSF: header e footer - esempio

```
 1. <?xml version="1.0"?>
 2.
 3. <!DOCTYPE faces-config PUBLIC
 4.   "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
 5.   "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">
 6.
 7. <faces-config>
 8.    <managed-bean>
 9.       <managed-bean-name>numberList</managed-bean-name>
10.       <managed-bean-class>java.util.ArrayList</managed-bean-class>
11.       <managed-bean-scope>session</managed-bean-scope>
12.       <list-entries>
13.          <value>1</value>
14.          <value>2</value>
15.          <value>3</value>
16.          <value>4</value>
17.          <value>5</value>
18.       </list-entries>
19.    </managed-bean>
20. </faces-config>
```

JSF: tabelle caricate da database

> Spesso i dati di una tabella sono rappresentati da liste dinamiche provenienti da DB



Customer ID	Name	Phone Number	Address	City	State
1	William Dupont	(652)488-9931	801 Oak Street	Eugene	Nebraska
2	Anna Keeney	(716)834-8772	86 East Amherst Street	Buffalo	New York
3	Mariko Randor	(451)842-8933	923 Maple Street	Springfield	Tennessee
4	John Wilson	(758)955-5934	8122 Genessee Street	El Reno	Oklahoma
5	Lynn Seckinger	(552)767-1935	712 Kehr Street	Kent	Washington
6	Richard Tattersall	(455)282-2936	21 South Park Drive	Dallas	Texas
7	Gabriella Sarintia	(819)152-8937	81123 West Seneca Street	Denver	Colorado
8	Lisa Hartwig	(818)852-1937	6652 Sheridan Drive	Sheridan	Wyoming
9	Shirley Jones	(992)488-3931	2831 Main Street	Butte	Montana
10	Bill Sprague	(316)962-0632	1043 Cherry Street	Cheektowaga	New York
11	Greg Doench	(136)692-6023	99 Oak Street	Upper Saddle River	New Jersey
12	Solange Nadeau	(255)767-0935	177 Rue St. Catherine	Montreal	Quebec
13	Heather McGann	(554)282-0936	7192 913 West Park	Buloxie	Mississippi
14	Roy Martin	(918)888-0937	5571 North Olean Avenue	White River	Arkansas
15	Claude Loubier	(857)955-0934	1003 Rue de la Montagne	St. Marguerite de Lingwick	Quebec
16	Dan Woodard	(703)555-1212	2993 Tonawonda Street	Springfield	Missouri
17	Ron Dunlap	(761)678-4251	5579 East Seneca Street	Kansas City	Kansas
18	Keith Frankart	(602)152-6723	88124 Milpidas Lane	Springfield	Maryland
19	Andre Nadeau	(541)842-0933	94219 Rue Florence	St. Marguerite de Lingwick	Quebec
20	Horace Celestin	(914)843-6553	99423 Spruce Street	Ann Arbor	Michigan

JSF: tabelle caricate da database - esempio

```
 1. <html>
 2.    <%@ taglib uri="http://java.sun.com/jsf/core"  prefix="f" %>
 3.    <%@ taglib uri="http://java.sun.com/jsf/html"  prefix="h" %>
 4.    <f:view>
 5.       <head>
 6.          <link href="styles.css" rel="stylesheet" type="text/css"/>
 7.          <f:loadBundle basename="messages" var="msgs"/>
 8.          <title>
 9.             <h:outputText value="#{msgs.pageTitle}"/>
10.          </title>
11.       </head>
12.       <body>
13.          <h:form>
14.             <h:dataTable value="#{customer.all}" var="customer"
15.                styleClass="customers"
16.                headerClass="customersHeader" columnClasses="custid,name">
17.                <h:column>
18.                   <f:facet name="header">
19.                      <h:outputText value="#{msgs.customerIdHeader}"/>
20.                   </f:facet>
21.                   <h:outputText value="#{customer.Cust_ID}"/>
22.                </h:column>
23.                <h:column>
24.                   <f:facet name="header">
25.                      <h:outputText value="#{msgs.nameHeader}"/>
26.                   </f:facet>
27.                   <h:outputText value="#{customer.Name}"/>
28.                </h:column>
```

JSF: tabelle caricate da database - esempio

```
29.                 <h:column>
30.                    <f:facet name="header">
31.                       <h:outputText value="#{msgs.phoneHeader}"/>
32.                    </f:facet>
33.                    <h:outputText value="#{customer.Phone_Number}"/>
34.                 </h:column>
35.                 <h:column>
36.                    <f:facet name="header">
37.                       <h:outputText value="#{msgs.addressHeader}"/>
38.                    </f:facet>
39.                    <h:outputText value="#{customer.Street_Address}"/>
40.                 </h:column>
41.                 <h:column>
42.                    <f:facet name="header">
43.                       <h:outputText value="#{msgs.cityHeader}"/>
44.                    </f:facet>
45.                    <h:outputText value="#{customer.City}"/>
46.                 </h:column>
47.                 <h:column>
48.                    <f:facet name="header">
49.                       <h:outputText value="#{msgs.stateHeader}"/>
50.                    </f:facet>
51.                    <h:outputText value="#{customer.State}"/>
52.                 </h:column>
53.              </h:dataTable>
54.           </h:form>
55.        </body>
56.     </f:view>
57. </html>
```

JSF: tabelle caricate da database - esempio

```
 1. package it.unibo.deis;
 2.
 3. import java.sql.Connection;
 4. import java.sql.ResultSet;
 5. import java.sql.SQLException;
 6. import java.sql.Statement;
 7. import javax.naming.Context;
 8. import javax.naming.InitialContext;
 9. import javax.naming.NamingException;
10. import javax.servlet.jsp.jstl.sql.Result;
11. import javax.servlet.jsp.jstl.sql.ResultSupport;
12. import javax.sql.DataSource;
13.
14. public class CustomerBean {
15.    private Connection conn;
16.
17.    public void open() throws SQLException, NamingException {
18.       if (conn != null) return;
19.       Context ctx = new InitialContext();
20.       DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/test");
21.       conn = ds.getConnection();
22.    }
```

JSF: tabelle caricate da database - esempio

```
23.     public Result getAll() throws SQLException, NamingException {
24.        try {
25.           open();
26.           Statement stmt = conn.createStatement();
27.           ResultSet result = stmt.executeQuery("SELECT * FROM Customers");
28.           return ResultSupport.toResult(result);
29.        } finally {
30.           close();
31.        }
32.     }
33.
34.     public void close() throws SQLException {
35.        if (conn == null) return;
36.        conn.close();
37.        conn = null;
38.     }
39. }
```

JSF: table models

> Il tag **`h:dataTable`** *wrappa* internamente i vari oggetti che rappresentano i dati da processare in un modello che estende **`javax.faces.model.DataModel`**.

> Le seguenti classi, una per ogni tipo supportato da **`h:dataTable`**, estendono **`javax.faces.model.DataModel`** :

- **`ArrayDataModel`**
- **`ListDataModel`**
- **`ResultDataModel`**
- **`ResultSetDataModel`**
- **`ScalarDataModel`**

E' comunque sempre possibile accedere agli oggetti *wrappati* utilizzando il metodo **`DataModel.getWrappedData()`**

JSF: modifiche di data table

> Vediamo un esempio di modifica della data table

JSF: modifiche di data table - esempio

```
 1. <html>
 2.    <%@ taglib uri="http://java.sun.com/jsf/core"  prefix="f" %>
 3.    <%@ taglib uri="http://java.sun.com/jsf/html"  prefix="h" %>
 4.    <f:view>
 5.       <head>
 6.          <link href="styles.css" rel="stylesheet" type="text/css"/>
 7.          <f:loadBundle basename="messages" var="msgs"/>
 8.          <title>
 9.             <h:outputText value="#{msgs.windowTitle}"/>
10.          </title>
11.       </head>
12.       <body>
13.          <h:form>
14.             <h:dataTable value="#{tableData.names}" var="name"
15.                styleClass="names" headerClass="namesHeader"
16.                columnClasses="last,first">
17.                <h:column rendered="#{tableData.editable}">
18.                   <f:facet name="header">
19.                      <h:outputText value="#{msgs.deleteColumnHeader}"/>
20.                   </f:facet>
21.                   <h:selectBooleanCheckbox value="#{name.markedForDeletion}"
22.                      onchange="submit()"/>
23.                </h:column>
```

JSF: modifiche di data table - esempio

```
24.                 <h:column>
25.                    <f:facet name="header">
26.                       <h:outputText value="#{msgs.lastColumnHeader}"/>
27.                    </f:facet>
28.                    <h:outputText value="#{name.last}"/>
29.                    <f:verbatim>,</f:verbatim>
30.                 </h:column>
31.                 <h:column>
32.                    <f:facet name="header">
33.                       <h:outputText value="#{msgs.firstColumnHeader}"/>
34.                    </f:facet>
35.                    <h:outputText value="#{name.first}"/>
36.                 </h:column>
37.              </h:dataTable>
38.              <p>
39.              <h:outputText value="#{msgs.editPrompt}"/>
40.              <h:selectBooleanCheckbox onchange="submit()"
41.                 value="#{tableData.editable}"/>
42.              <h:commandButton value="#{msgs.deleteButtonText}"
43.                 rendered="#{tableData.editable}"
44.                 action="#{tableData.deleteNames}"
45.                 disabled="#{not tableData.anyNamesMarkedForDeletion}"/>
46.           </h:form>
47.        </body>
48.     </f:view>
49. </html>
```

JSF: modifiche di data table - esempio

```
 1. package it.unibo.deis;
 2.
 3. public class Name {
 4.    private String first;
 5.    private String last;
 6.    private boolean markedForDeletion = false;
 7.
 8.    public Name(String first, String last) {
 9.       this.first = first;
10.       this.last = last;
11.    }
12.
13.    public void setFirst(String newValue) { first = newValue; }
14.    public String getFirst() { return first; }
15.
16.   public void setLast(String newValue) { last = newValue; }
17.   public String getLast() { return last; }
18.
19.   public boolean isMarkedForDeletion() { return markedForDeletion; }
20.   public void setMarkedForDeletion(boolean newValue) {
21.      markedForDeletion = newValue;
22.   }
23.}
```

JSF: modifiche di data table - esempio

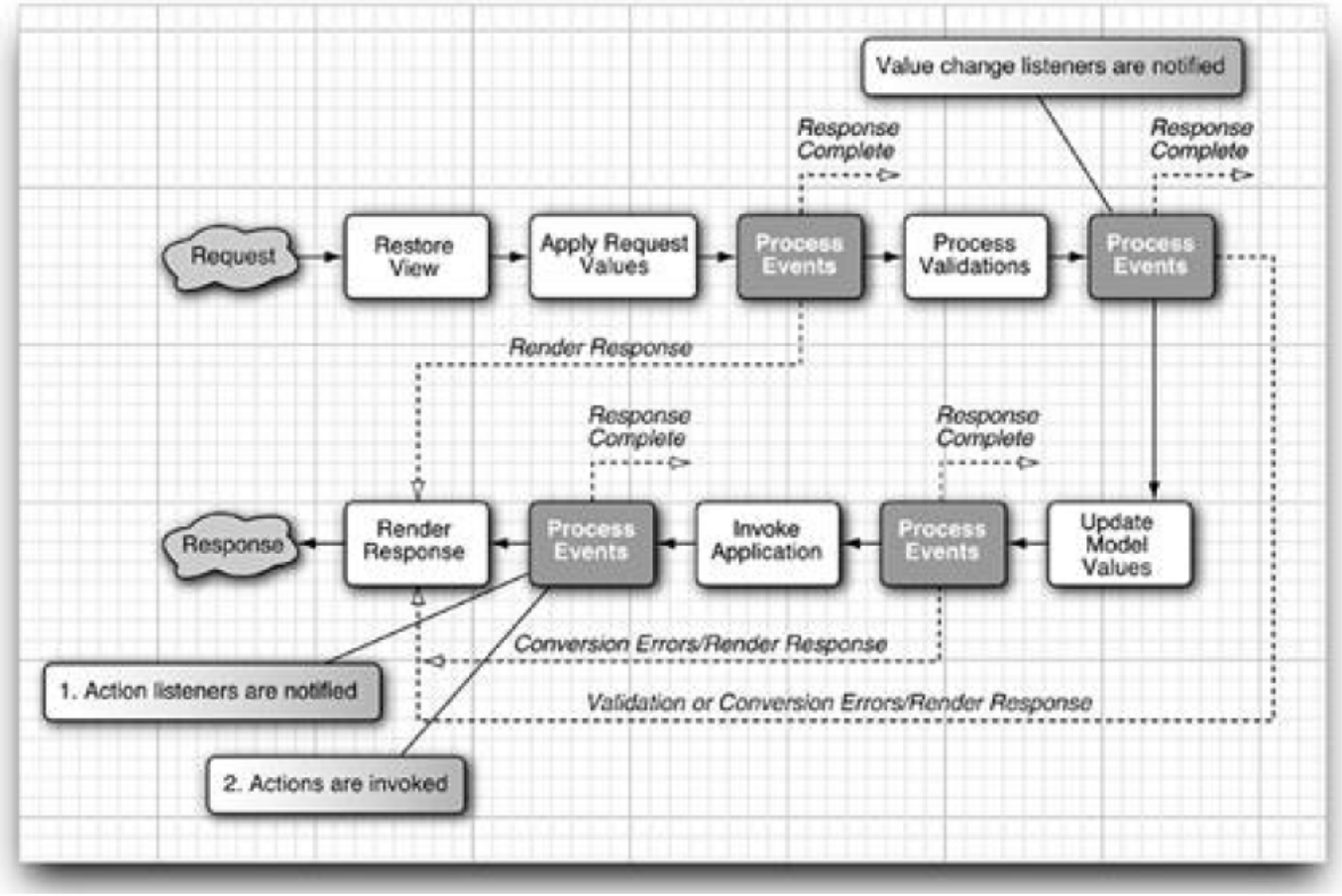
```
 1. package it.unibo.deis;
 2.
 3. import javax.faces.model.DataModel;
 4. import javax.faces.model.ArrayDataModel;
 5.
 6. public class TableData {
 7.    private boolean editable = false;
 8.    private ArrayDataModel model = null;
 9.
10.    private static final Name[] names = {
11.       new Name("Anna", "Keeney"),
12.       new Name("John", "Wilson"),
13.       new Name("Mariko", "Randor"),
14.       new Name("William", "Dupont"),
15.    };
16.
17.    public TableData() { model = new ArrayDataModel(names); }
18.
19.    public DataModel getNames() { return model; }
20.
21.    public boolean isEditable() { return editable; }
22.    public void setEditable(boolean newValue) { editable = newValue; }
23.
24.    public String deleteNames() {
25.       if (!getAnyNamesMarkedForDeletion())
26.          return null;
27.
28.       Name[] currentNames = (Name[]) model.getWrappedData();
29.       Name[] newNames = new Name[currentNames.length
30.          - getNumberOfNamesMarkedForDeletion()];
```

JSF: modifiche di data table - esempio

```
32.         for(int i = 0, j = 0; i < currentNames.length; ++i) {
33.            Name name = (Name) currentNames[i];
34.            if (!name.isMarkedForDeletion()) {
35.               newNames[j++] = name;
36.            }
37.         }
38.         model.setWrappedData(newNames);
39.         return null;
40.      }
41.      public int getNumberOfNamesMarkedForDeletion() {
42.         Name[] currentNames = (Name[]) model.getWrappedData();
43.         int cnt = 0;
44.
45.         for(int i = 0; i < currentNames.length; ++i) {
46.            Name name = (Name) currentNames[i];
47.            if (name.isMarkedForDeletion())
48.               ++cnt;
49.         }
50.         return cnt;
51.      }
52.      public boolean getAnyNamesMarkedForDeletion() {
53.         Name[] currentNames = (Name[]) model.getWrappedData();
54.         for(int i = 0; i < currentNames.length; ++i) {
55.            Name name = (Name) currentNames[i];
56.            if (name.isMarkedForDeletion())
57.               return true;
58.         }
59.         return false;
60.      }
61. }
```

JSF: gestione eventi

> Riprendiamo lo schema di processamento della request (lifecycle) da parte del framework JSF



JSF: gestione eventi

> Come abbiamo già detto il lifecycle è definito da un insieme preciso di fasi.

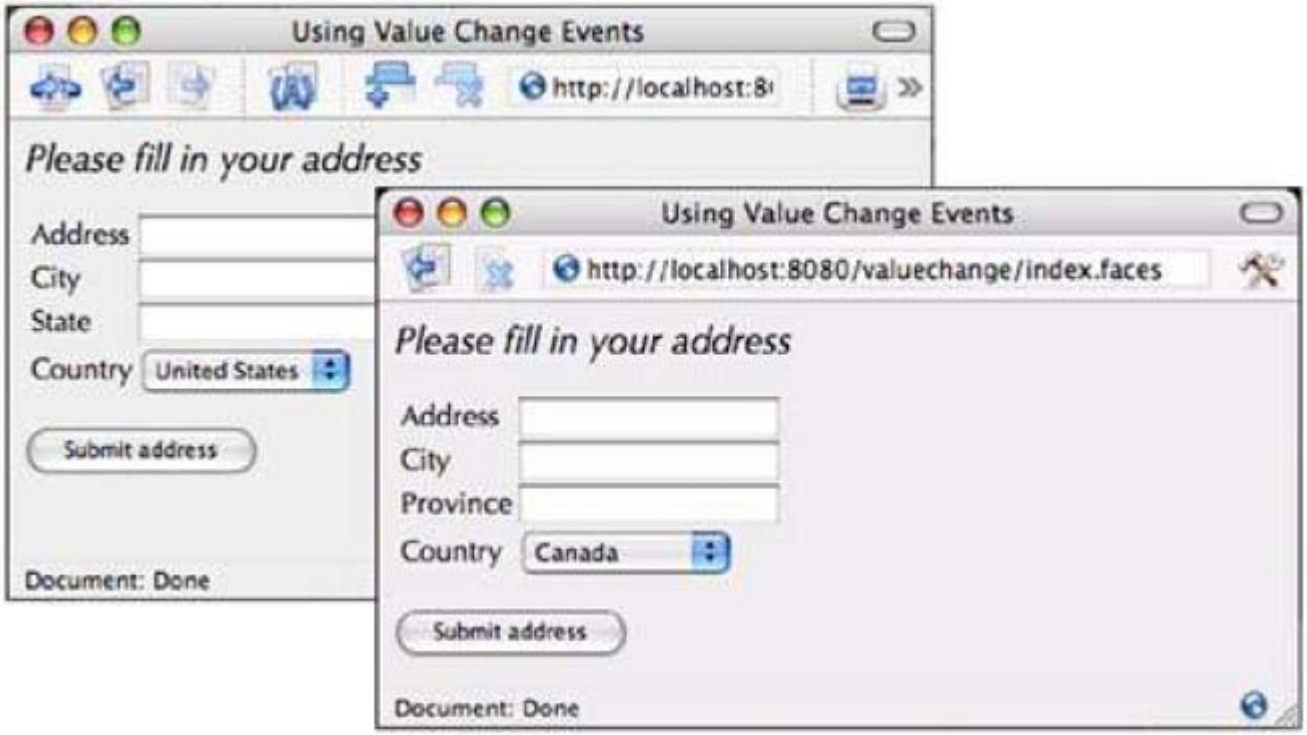
> Le fasi sono le seguenti :

- **Restore View**
- **Apply Request Value**
- **Process Validations**
- **Update Model Value**
- **Invoke Application**
- **Render Response**

La fase di Restore View recupera il *component tree* della vista. La fase di Apply Request Value riporta i parametri della richiesta nei *submitted values* dei corrispondenti componenti. La fase di Process Validations converte i valori nei tipi opportuni, li valida e aggiorna i *local values* dei componenti. La fase di Update Model Value copia i valori validati nel modello. La fase di Invoke Application notifica *action listener* e *action* (nell'ordine indicato). La fase di Render Response salva lo stato e carica la *view* successiva.

JSF: Value Change Event

> Una categoria di eventi definita dalle specifiche JSF è detta *Value Change Event*. Questo evento viene generato (a fronte di un'interazione dell'utente con un controllo HTML di input) durante la fase di *Apply Request Value* e inserito in una coda eventi. Successivamente, dopo la fase di *Process Validations*, l'evento viene inviato in *broadcast* ai vari listener registrati. Gli eventi vengono sollevati da componenti di input



JSF: Value Change Event - esempio

```
 1. <html>
 2.    <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 3.    <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
 4.    <f:view>
 5.       <link href="styles.css" rel="stylesheet" type="text/css"/>
 6.       <f:loadBundle basename="messages" var="msgs"/>
 7.        <head>
 8.         <title>
 9.             <h:outputText value="#{msgs.windowTitle}"/>
10.          </title>
11.       </head>
12.
13.       <body>
14.          <h:outputText value="#{msgs.pageTitle}" styleClass="emphasis"/>
15.          <p/>
16.          <h:form>
17.             <h:panelGrid columns="2">
18.                <h:outputText value="#{msgs.streetAddressPrompt}"/>
19.                <h:inputText value="#{form.streetAddress}" id="streetAddress"/>
20.
21.                <h:outputText value="#{msgs.cityPrompt}"/>
22.                <h:inputText value="#{form.city}"/>
23.
24.                <h:outputText value="#{msgs.statePrompt}"/>
25.                <h:inputText value="#{form.state}"/>
26.                <h:outputText value="#{msgs.countryPrompt}"/>
27.
28.                <h:selectOneMenu value="#{form.country}"
29.                   onchange="submit()"
30.                   valueChangeListener="#{form.countryChanged}">
31.                   <f:selectItems value="#{form.countryNames}"/>
32.                </h:selectOneMenu>
33.             </h:panelGrid>
34.             <p/>
35.             <h:commandButton value="#{msgs.submit}"/>
36.          </h:form>
37.       </body>
38.    </f:view>
39. </html>
```

JSF: Value Change Event - esempio

```
 1. package it.unibo.deis;
 2.
 3. import java.util.ArrayList;
 4. import java.util.Collection;
 5. import java.util.Locale;
 6. import javax.faces.context.FacesContext;
 7. import javax.faces.event.ValueChangeEvent;
 8. import javax.faces.model.SelectItem;
 9.
10. public class RegisterForm {
11.    private String streetAddress;
12.    private String city;
13.    private String state;
14.    private String country;
15.
16.    private static final String US = "United States";
17.    private static final String CANADA = "Canada";
18.    private static final String[] COUNTRY_NAMES = { US, CANADA };
19.    private static ArrayList countryItems = null;
20.
21.    // PROPERTY: countryNames
22.    public Collection getCountryNames() {
23.       if(countryItems == null) {
24.          countryItems = new ArrayList();
25.          for (int i = 0; i < COUNTRY_NAMES.length; ++i) {
26.             countryItems.add(new SelectItem(COUNTRY_NAMES[i]));
27.          }
28.       }
29.       return countryItems;
30.    }
```

JSF: Value Change Event - esempio

```
31.
32.     // PROPERTY: streetAddress
33.     public void setStreetAddress(String newValue) { streetAddress = newValue; }
34.     public String getStreetAddress() { return streetAddress; }
35.
36.     // PROPERTY: city
37.     public void setCity(String newValue) { city = newValue; }
38.     public String getCity() { return city; }
39.
40.     // PROPERTY: state
41.     public void setState(String newValue) { state = newValue; }
42.     public String getState() { return state; }
43.
44.     // PROPERTY: country
45.     public void setCountry(String newValue) { country = newValue; }
46.     public String getCountry()              { return country; }
47.
48.     public void countryChanged(ValueChangeEvent event) {
49.        FacesContext context = FacesContext.getCurrentInstance();
50.
51.        if(US.equals((String) event.getNewValue()))
52.           context.getViewRoot().setLocale(Locale.US);
53.        else
54.           context.getViewRoot().setLocale(Locale.CANADA);
55.     }
56. }
```

JSF: javax.faces.event.ValueChangeEvent

> Il parametro di input rivevuto dal metodo notificato dal framework JSF è di tipo **`javax.faces.event.ValueChangeEvent`**. Questa classe fornisce i seguenti metodi:

- **`UIComponent getComponent()`**
- **`Object getNewValue()`**
- **`Object getOldValue()`**

> La classe estende **`javax.faces.event.FacesEvent`**. Questa classe astratta presenta, tra gli altri, i seguenti metodi (vedere API per le specifiche):

- **`void queue()`**
- **`PhaseId getPhaseId()`**
- **`void setPhaseId(PhaseId)`**

JSF: Action Event

> Un'altra categoria di eventi definita dalle specifiche JSF è detta *Action Event*. Anche questi eventi vengono generati durante la fase di *Apply Request Value* (a seguito dell'interazione dell'utente con un controllo HTML che provoca il *submit* del FORM) e inseriti in una opportuna coda eventi associata al componente che li ha generati. Successivamente, durante la fase di *Invoke Application*, JSF invoca i vari listener registrati preposti alla gestione dell'evento. Gli eventi sono prodotti da componenti di tipo command (bottoni e link) quando questi sono attivati (e, quindi, quando il corrispondente valore è presente nella *HTTP request*). Questi componenti sono anche detti *action sources*.

> JSF distingue tra *action listener* e *action*. Concettualmente le *action* sono pensate per gestire la business logic e partecipare al processo di *navigation handling* (non hanno nozione di concetti di UI). *Action listener* sono pensati per gestire la logica di UI (logica di presentazione). Spesso i listener associati a questi eventi lavorano di concerto (per esempio quando un *action* necessita di informazioni di UI).

JSF: Action Event

> Vediamo ora un esempio. Quando si effettua un click su una parte specifica dell'immagine si determina (attraverso le coordinate inviate come *HTTP parameter*) quale dettaglio presentare nella pagina successiva. La *action* non ha conoscenza circa i dettagli di UI, necessita quindi di collaborare con *action listener*

JSF: Action Event - esempio

```
 1. <html>
 2.    <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 3.    <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
 4.    <f:view>
 5.       <head>
 6.          <link href="styles.css" rel="stylesheet" type="text/css"/>
 7.          <f:loadBundle basename="messages" var="msgs"/>
 8.          <title>
 9.             <h:outputText value="#{msgs.windowTitle}"/>
10.          </title>
11.       </head>
12.       <body>
13.          <h:form style="text-align: center">
14.             <h:commandButton image="mountrushmore.jpg"
15.                actionListener="#{rushmore.listen}"
16.                action="#{rushmore.act}"/>
17.          </h:form>
18.       </body>
19.    </f:view>
20. </html>
```

JSF: Action Event - esempio

```
 1. package it.unibo.deis;
 2.
 3. import java.awt.Point;
 4. import java.awt.Rectangle;
 5. import java.util.Map;
 6. import javax.faces.context.FacesContext;
 7. import javax.faces.event.ActionEvent;
 8.
 9. public class Rushmore {
10.    private String outcome = null;
11.    private Rectangle washingtonRect = new Rectangle(70, 30, 40, 40);
12.    private Rectangle jeffersonRect = new Rectangle(115, 45, 40, 40);
13.    private Rectangle rooseveltRect = new Rectangle(135, 65, 40, 40);
14.    private Rectangle lincolnRect = new Rectangle(175, 62, 40, 40);
15.
16.    public void listen(ActionEvent e) {
17.       FacesContext context = FacesContext.getCurrentInstance();
18.       String clientId = e.getComponent().getClientId(context);
19.       Map requestParams =
         context.getExternalContext().getRequestParameterMap();
20.
21.  int x = new Integer((String) requestParams.get(clientId + ".x"))
22.             .intValue();
```

JSF: Action Event - esempio

```
23.         int y = new Integer((String) requestParams.get(clientId + ".y"))
24.               .intValue();
25.
26.         outcome = null;
27.
28.         if (washingtonRect.contains(new Point(x, y)))
29.            outcome = "washington";
30.
31.         if (jeffersonRect.contains(new Point(x, y)))
32.            outcome = "jefferson";
33.
34.         if (rooseveltRect.contains(new Point(x, y)))
35.            outcome = "roosevelt";
36.
37.         if (lincolnRect.contains(new Point(x, y)))
38.            outcome = "lincoln";
39.      }
40.
41.      public String act() {
42.         return outcome;
43.      }
44.   }
```

JSF: tag Event Listener

> Sinora abbiamo aggiunto *action* e *value change* listener ai componenti usando gli attributi **`actionListener`** e **`valueChangeListener`**, rispettivamente. E' possibile ottenere lo stesso risultato utilizzando i tag **`f:actionListener`** e **`f:valueChangeListener`**. Tipicamente si usano i tag quando c'è la necessità di registrare multipli *action listener* e *value change listener* per un singolo componente

> I due markup sottostanti consentono di ottenere lo stesso risultato:

```
<h:selectOneMenu value="#{form.country}" onchange="submit()"
    valueChangeListener="#{form.countryChanged}">
    <f:selectItems value="#{form.countryNames}"/>
</h:selectOneMenu>

<h:selectOneMenu value="#{form.country}" onchange="submit()">
    <f:valueChangeListener type=" it.unibo.deis.CountryListener"/>
    <f:selectItems value="#{form.countryNames}"/>
</h:selectOneMenu>
```

JSF: tag Event Listener

> Si noti che nel primo caso utilizziamo un *method binding* per registrare il *listener*, espresso con JSF EL; nel secondo una classe Java, indicata come valore dell'attributo **`type`** del tag. Tale classe deve implementare l'interfaccia **`javax.faces.event.ValueChangeListener`**. Esempio:

```
public class CountryListener implements ValueChangeListener {
    private static final String US = "United States";

    public void processValueChange(ValueChangeEvent event) {
      FacesContext context = FacesContext.getCurrentInstance();

      if (US.equals((String) event.getNewValue()))
          context.getViewRoot().setLocale(Locale.US);
      else
          context.getViewRoot().setLocale(Locale.CANADA);
   }
}
```

> L'interfaccia specifica il metodo **`processValueChange`**, che riceve in input un parametro **`ValueChangeEvent`** e non ritorna parametri

JSF: tag Event Listener

> Discorso assolutamente analogo per *action listener*. Ecco i due markup equivalenti:

```
<h:commandButton image="mountrushmore.jpg"
    actionListener="#{rushmore.listen}"
    action="#{rushmore.act}"/>

<h:commandButton image="mountrushmore.jpg" action="#{rushmore.act}">
    <f:actionListener type="it.unibo.deis.RushmoreListener"/>
</h:commandButton>
```

> L'interfaccia da implementare dalla classe che rappresenta il *listener* è **`javax.faces.event.ActionListener`**. La firma dell'unico metodo specificato dall'interfaccia è il seguente:

```
void processAction(ActionEvent)
```

> Se si specificano più listener per un componente, essi sono invocati nel seguente ordine: dapprima quello specificato dall'attributo, successivamente quelli indicati dai tag, nell'ordine in cui sono dichiarati

JSF: componenti `immediate`

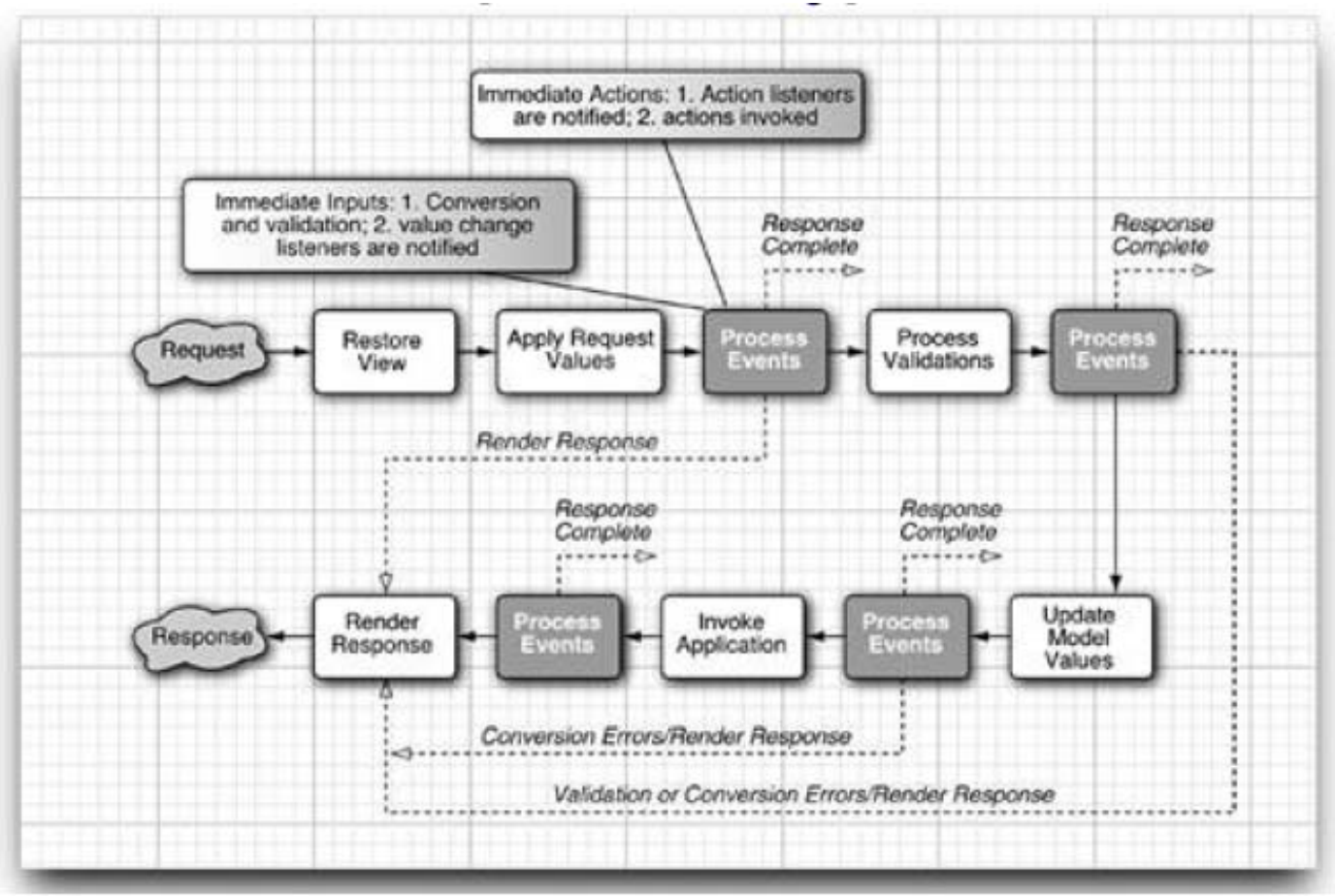
> I componenti di tipo *command* e *input* prevedono l'attributo **`immediate`** che assume valori booleani.

> Se il corrispondente valore è *true*, per i componenti di *input* gli step di "conversione" e "validazione" vengono eseguiti immediatamente dopo la fase di *Apply Request Value* e successivamente vengono notificati i listener degli eventi *Value Change*. Si passa poi alla fase di *Render Response* oppure, nel caso si produca –per esempio- un contenuto binario, si completa la risposta manipolando direttamente lo *stream* di *output*.

> Per gli *action sources*, la valorizzazione a *true* di **`immediate`** determina l'invocazione di *action listener* e *action* (nell'ordine) dopo la fase di *Apply Request Value*, saltando le altre fasi e passando poi direttamente alla fase di *Render Response*, a meno che non si scriva direttamente sullo *stream* di *output* e si completi la risposta.

JSF: componenti `immediate`

> Nella figura sottostante viene illustrato graficamente quanto detto



JSF: componenti di input `immediate`

> Riprendiamo un esempio visto precedentemente, quello relativo al cambio della lingua. Supponiamo di aggiungere un semplice attributo *required* al campo di indirizzo: la selezione del paese dal menu determina la segnalazione di errore che, in questo caso, non è il comportamento desiderato (la validazione dovrebbe avvenire al *submit* dell'intero FORM)

JSF: componenti di input `immediate`

> La soluzione passa per la trasformazione di menu country in un componente *immediate*. I componenti di input *immediate* effettuano conversione e validazione e successivamente sollevano gli eventi *Value Change* all'inizio del lifecycle, dopo *Apply Request Value*, invece che dopo *Process Validations*.

```
<h:selectOneMenu value="#{form.country}" onchange="submit()"
immediate="true" valueChangeListener="#{form.countryChanged}">
    <f:selectItems value="#{form.countryNames}"/>
</h:selectOneMenu>
```

> Tuttavia, questo accorgimento (da solo) non consente di evitare la validazione degli altri componenti che, infatti, avverrebbe comunque. È necessario, per scongiurare il comportamento indesiderato evidenziato nella pagina precedente, invocare esplicitamente la fase di *Render Response* alla fine del *Value Change listener*. In questo caso si determina un cambiamento nel lifecycle che consente di superare il problema

JSF: componenti command `immediate`

> Per quanto attiene i componenti di tipo command *(action sources*), è sufficiente limitarsi al setting dell'attributo *immediate.*

```
<h:commandLink actionListener="#{localeChanger.changeLocale}"
immediate="true">
    <h:graphicImage value="/german_flag.gif" style="border: 0px"/>
    <f:param name="locale" value="german"/>
</h:commandLink>
```

> I listener registrati sugli eventi da essi generati, infatti, indipendentemente dal fatto che i componenti siano o meno *immediate*, passano sempre alla fase di *Render Response*, eliminando così alla radice il probema.