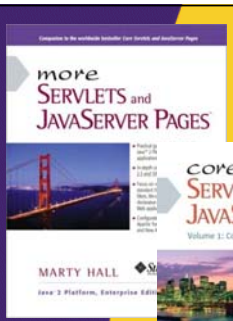
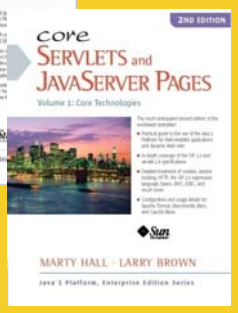
JSF 2 Composite Components
Part III – Backing Components

JSF 2.2 Version

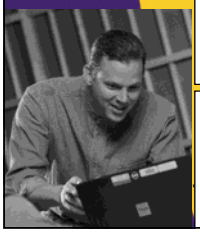
Originals of slides and source code for examples: http://www.coreservlets.com/JSF-Tutorial/jsf2/
Also see the PrimeFaces tutorial – http://www.coreservlets.com/JSF-Tutorial/primefaces/
and customized JSF2 and PrimeFaces training courses – http://courses.coreservlets.com/jsf-training.html





For live training on JSF 2, PrimeFaces, or other Java EE topics, email hall@coreservlets.com

Marty is also available for consulting and development support



Taught by the author of *Core Servlets and JSP*, this tutorial, and JSF 2.2 version of *Core JSF*. 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, Ajax, jQuery, Spring MVC, JSP, Android, general Java, Java 8 lambdas/streams, GWT, custom topic mix
  - Courses available in any location worldwide. Maryland/DC area companies can also choose afternoon/evening courses.
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Hadoop, Spring, Hibernate/JPA, RESTful Web Services

Contact hall@coreservlets.com for details

Topics in This Section

- **Backing components with custom parsing**
  - Extend UIInput and implement NamingContainer
  - Declare with @FacesComponent
  - Use encodeBegin and getConvertedValue
- **Backing components with setter methods called inside component code**
- **Ajaxified components**

Review: Input Components

Input Components

- **Idea**
  - A composite component that will go inside h:form and be used to gather input from the user
- **Features**
  - Must be associated with a bean value
    - Use “type” attribute to enforce the type of the value
  - Often has its own style sheet
    - Load with h:outputStylesheet
  - May need to assemble final value out of submitted pieces
  - May need to call component setter methods based on attribute values
  - May have embedded Ajax behavior
  - May be built out of a rich JavaScript GUI element

Backing Components: Customizing Parsing of Submitted Values

Backing Components: Idea

- **No backing components**
  - If each input element corresponds to an individual bean property
    - In this case, the situation is simple: just use “value” of h:inputText or other input element and point at bean property
    - You can use custom converter if textfield value needs to be parsed to become bean value
- **Backing components**
  - If multiple input elements need to be combined to produce a single bean property
    - In this case, make a class that extends UIInput and implements NamingContainer. Declare with @FacesComponent. Use encodeBegin and getConvertedValue.

Backing Components: Code Summary

- **Java code**

```
@FacesComponent("foo.Bar")
public class MyComponent extends UIInput
                         implements NamingContainer {
  Override methods, esp. encodeBegin & getConvertedValue
}
```

- **Composite component**

```
<cc:interface componentType="foo.Bar">
  …
</cc:interface>
<cc:implementation>
  …
</cc:implementation>
```

Backing Component: Java Code

- **@FacesComponent(…)**
  - You usually give the package and name of current class, but technically, this can be any string
  - Must match what is used for “componentType” in cc:interface
- **getFamily**
  - return("javax.faces.NamingContainer");
- **getSubmittedValue**
  - return(this);

Backing Component: Java Code (Continued)

- **encodeBegin**
  - Purpose: to populate elements in input form so that initial display corresponds to current bean value
  - Find individual input elements with findComponent("id-used-in-composite-component")
  - Populate them by calling setValue
  - Call super.encodeBegin at end of method
- **getConvertedValue**
  - Purpose: to take individual submitted values and combine them into a single bean value
  - Find individual input elements with findComponent("id-used-in-composite-component")
  - Find their values with getSubmittedValue
  - Make an Object and return it from method

Example: inputDate1

- **Idea**
  - Collect a day, month, and year.
  - Have it correspond to a Date
- **Code summary**
  - Java class
    - Make Java class that extends UIInput and implements NamingContainer
    - encodeBegin determines the day, month and year of the current Date, and sets the menus appropriately
    - getConvertedValue finds the submitted day, month, and year, and creates a Date object corresponding to them
  - Composite component
    - Use three h:selectOneMenu elements. Give each an id.
    - The string of @FacesComponent from Java class is used for the componentType of cc:interface

Java Class (Beginning)

```
@FacesComponent("coreservlets.DateComponent1")
public class DateComponent1 extends UIInput
                            implements NamingContainer {
  @Override
  public String getFamily() {
    return("javax.faces.NamingContainer");
  }

  @Override
  public Object getSubmittedValue() {
    return(this);
  }
```

This string will exactly match the componentType given in cc:implementation of the component.

These return values almost never change.

Note: Code for this class adapted from the date class in Chapter 9 of *Core JavaServer Faces, 3rd Edition* by David Geary and Cay Horstmann.

Java Class (Continued)

```
@Override
public void encodeBegin(FacesContext context)
    throws IOException {
  Date date = (Date)getValue();
  Calendar cal = new GregorianCalendar();
  cal.setTime(date);
  UIInput dayComponent = (UIInput)findComponent("day");
  UIInput monthComponent = (UIInput)findComponent("month");
  UIInput yearComponent = (UIInput)findComponent("year");
  dayComponent.setValue(cal.get(Calendar.DATE));
  monthComponent.setValue(cal.get(Calendar.MONTH) + 1);
  yearComponent.setValue(cal.get(Calendar.YEAR));
  super.encodeBegin(context);
}
```

This is the id for the h:selectOneMenu, exactly as given in the composite component file. You don't have to construct the full HTML id, as we will have to do later when wrapping JavaScript components.

Java Class (Continued)

```
@Override
protected Object getConvertedValue(FacesContext context,
                                   Object newSubmittedValue)
    throws ConverterException {
  UIInput dayComponent = (UIInput)findComponent("day");
  UIInput monthComponent = (UIInput)findComponent("month");
  UIInput yearComponent = (UIInput)findComponent("year");
  int day =
    Integer.parseInt((String)dayComponent.getSubmittedValue());
  int month =
    Integer.parseInt((String)monthComponent.getSubmittedValue());
  int year =
    Integer.parseInt((String)yearComponent.getSubmittedValue());
  return(new GregorianCalendar(year, month-1, day).getTime());
}
```

Again, this is the id for the h:selectOneMenu, exactly as given in the composite component file. You don't have to construct the full HTML id, as we will have to do later when wrapping JavaScript components.

Managed Bean that Gives Choices for the Date Menus

```
@ManagedBean
@ApplicationScoped
public class DateChoices {
  private int[] days;
  private int[] years;
  private Map<String, Integer> months;

  public DateChoices() {
    days = DateUtils.intArray(1, 31);
    years = DateUtils.intArray(1900, 2100);
    months = new LinkedHashMap<String, Integer>();
    String[] names = new DateFormatSymbols().getMonths();
    for (int i = 0; i < 12; i++) {
      months.put(names[i], i + 1);
    }
  }
```

This will list 1 to 31 as the choices for the day in every month, including months that have less than 31 days. We will fix this later when we Ajaxify the component..

This gives the names of the months (January, etc.), but since the underlying data structure is a map, the associated value (Integer) will be returned as the user selection.

Note: Code for this class adapted from the Dates class in Chapter 9 of *Core JavaServer Faces, 3rd Edition* by David Geary and Cay Horstmann.

Managed Bean that Gives Choices for the Dates (Cont.)

```
  public int[] getDays() {
    return(days);
  }

  public int[] getYears() {
    return(years);
  }

  public Map<String, Integer> getMonths() {
    return(months);
  }
}
```

Helper Class

```
public class DateUtils {
  public static int[] intArray(int from, int to) {
    int[] nums = new int[to - from + 1];
    for (int i = 0; i < nums.length; i++) {
      nums[i] = from++;
    }
    return (nums);
  }

  // Several other Date-related utilities shown later

  private DateUtils() {} // Uninstantiatable class
}
```


Component File: Interface Section

```
<cc:interface
        componentType="coreservlets.DateComponent1">
  <cc:attribute name="value"
                type="java.util.Date"/>
</cc:interface>
```

This matches what was given in the @FacesComponent annotation (which is conventionally the fully qualified class name, but can be anything that you want).

Component File: Implementation Section

```
<cc:implementation>
  <h:selectOneMenu id="day"
                   converter="javax.faces.Integer">
    <f:selectItems value="#{dateChoices.days}"/>
  </h:selectOneMenu>
  <h:selectOneMenu id="month"
                   converter="javax.faces.Integer">
    <f:selectItems value="#{dateChoices.months}"/>
  </h:selectOneMenu>
  <h:selectOneMenu id="year"
                   converter="javax.faces.Integer">
    <f:selectItems value="#{dateChoices.years}"/>
  </h:selectOneMenu>
</cc:implementation>
```

These are the ids (exactly as given here) that you use with findComponent in encodeBegin and getConvertedValue.

Example Usage: Reservations at the JSF Resort

- **Idea**
  - Let user make a hotel reservation
- **Managed Bean**
  - Implements Nameable (for collecting the name)
  - Has two bean properties of type Date (getStartDate, setStartDate, getEndDate, setEndDate)
    - Enforces that checkout date is after checkin date
- **Page that uses component**
  - Collects name, checkin (arrival) date, and checkout (departure) date
- **Results page**
  - Shows confirmation with first name, last name, arrival date, and departure date

Managed Bean (Beginning)

```
@ManagedBean
public class ResortBean implements Nameable {
  private String firstName, lastName;
  private Date startDate=DateUtils.nextDay(new Date()),
               endDate=DateUtils.nextDay(startDate);
  public String getFirstName() {
    return(firstName);
  }
  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }
  public String getLastName() {
    return(lastName);
  }
  public void setLastName(String lastName) {
    this.lastName = lastName;
  }
```

Minimum stay is one day.

Check in tomorrow at earliest.

Managed Bean (Continued)

```
  public Date getStartDate() {
    return(startDate);
  }

  public void setStartDate(Date startDate) {
    this.startDate = startDate;
  }

  public Date getEndDate() {
    return(endDate);
  }

  public void setEndDate(Date endDate) {
    this.endDate = endDate;
  }
```

The startDate and endDate bean properties are of type Date. Automatically converted by the backing component.

Managed Bean (Continued)

```
public String getStartDay() {
  return(formatDate(startDate));
}

private String formatDate(Date date) {
  if (date == null) {
    return("");
  } else {
    return(String.format("%tA, %tB %te, %tY",
                         date, date, date, date));
  }
}

public String getEndDay() {
  return(formatDate(endDate));
}
```

Given a Date, returns a String "Day, Month Number, Year", e.g. "Wednesday, November 2, 2011". For results page.

Managed Bean (Continued)

```
public String register() {
  FacesContext context = FacesContext.getCurrentInstance();
  if (!startDate.before(endDate)) {
    endDate = DateUtils.nextDay(startDate);
    FacesMessage errorMessage =
      new FacesMessage("End date must be after start date");
    context.addMessage("registrationForm:checkoutDate",
                       errorMessage);
    return(null);
  } else {
    return("show-dates");
  }
}
}
```

This is the action controller method.

Helper Class

```
public class DateUtils {
  public static long millisInDay() {
    return(24*60*60*1000);
  }

  public static Date nextDay(Date date) {
    return(new Date(date.getTime()
                    + millisInDay()));
  }

  // Several other Date-related utilities shown elsewhere
}
```

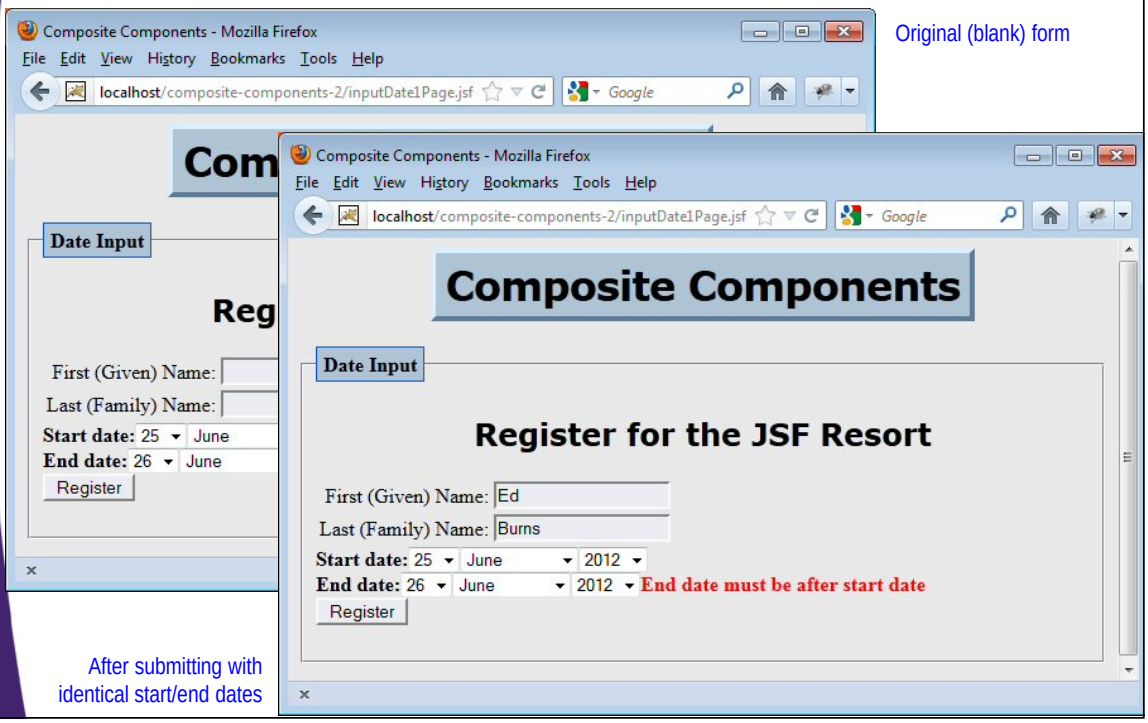

Facelets Page that Uses Component

```
...
<h:form id="registrationForm">
   <h2>Register for the JSF Resort</h2>
   <utils:inputName2 value="#{resortBean}"
                     firstNamePrompt="First (Given) Name"
                     lastNamePrompt="Last (Family) Name"/>
   <b>Start date:</b>
   <utils:inputDate1 value="#{resortBean.startDate}"/><br/>
   <b>End date:</b>
   <utils:inputDate1 value="#{resortBean.endDate}"
                     id="checkoutDate"/>
   <h:message for="registrationForm:checkoutDate"
              styleClass="error"/><br/>
   <h:commandButton action="#{resortBean.register}"
                    value="Register"/><p/>
</h:form>
...
```

Results Page

```
...
<h1 class="title">Reservation Confirmed</h1>
<p/>
<ul class="aligned">
  <li><b>Name:</b> #{resortBean.firstName}
                   #{resortBean.lastName}
  </li>
  <li><b>Arrival:</b> #{resortBean.startDay}</li>
  <li><b>Departure:</b> #{resortBean.endDay}</li>
</ul>
...
```

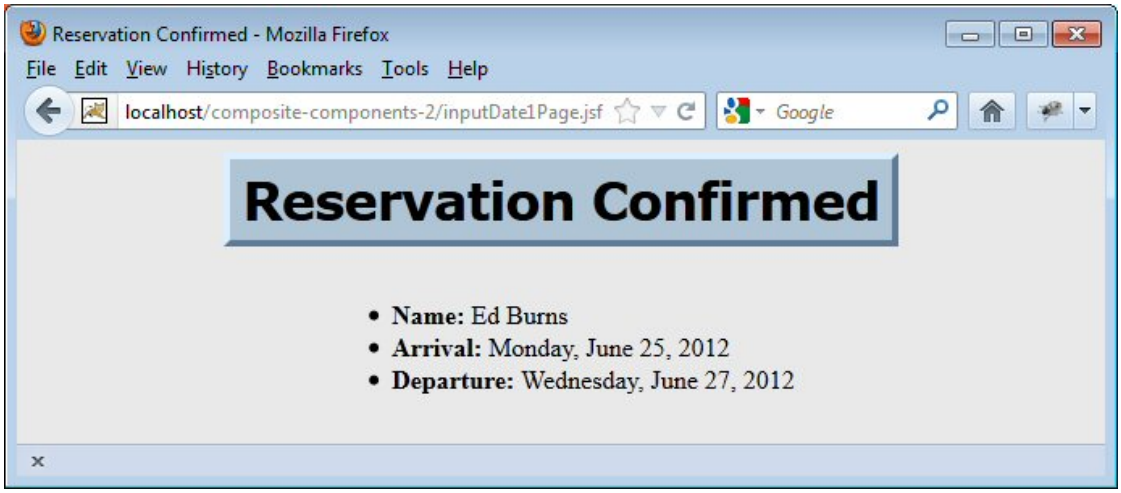

Result: Page that Uses Component



Original (blank) form

After submitting with identical start/end dates

Result: Results Page



Calling Setter Methods of Backing Components

Setter Methods: Idea

- **No setter methods in component**
  - Original input elements always look the same
    - In last example, the range of years is always the same

```
<h:selectOneMenu id="year" converter="javax.faces.Integer">
  <f:selectItems value="#{dateChoices.years}"/>
</h:selectOneMenu>
```

- **Setter methods in component**
  - Original input elements should be customizable
    - Range of years should be customizable by page author
    - But, this range needs to affect what is returned to f:selectItems. So, put methods in the component itself. Call setter methods. Then, make year range variable:

```
<h:selectOneMenu id="year" converter="javax.faces.Integer">
  <f:selectItems value="#{cc.years}"/>
</h:selectOneMenu>
```


Backing Components: Code Summary

- **Java code**

```
@FacesComponent(…)
public class MyComponent extends UIInput
                         implements NamingContainer {
  public void setStartYear(int startYear) {...}
  public void setEndYear(int startYear) {...}
}
```

- **Composite component**

```
<cc:implementation>
  <c:set target="#{cc}" property="startYear"
         value="#{cc.attrs.startYear}"/>
  <c:set target="#{cc}" property="endYear"
         value="#{cc.attrs.endYear}"/>
  …
</cc:implementation>
```

Calling Setter Methods from Expression Language: Options

- **Use c:set from JSTL**
  - Simple
  - Works in all JSF implementations
    - This is the approach used in these examples
- **Define a JSTL function**
  - Complex
  - Works in all JSF implementations
- **Call directly from expression language**
  - Simplest of all
  - Works only in Java EE 6
    - Passing arguments to methods was covered in the tutorial section on the expression language

Using c:set from JSTL

- **Load the JSTL core library**

```
<html xmlns="http://www.w3.org/1999/xhtml"
      …
      xmlns:c="http://xmlns.jcp.org/jsp/jstl/core">
```

- **Call the setter method**
  - General

```
<c:set target="#{somethingThatEvaluatesToaBean}"
       property="beanPropertyName"
       value="#{someValue}"/>
```

  - Usual way in composite components

```
<c:set target="#{cc}"
       property="beanPropertyName"
       value="#{cc.attrs.someAttributeValue}"/>
```

Reminder: the bean property name is the name of the setter method, minus the word "set", with the next letter changed to lower case.

Example: inputDate2

- **Idea**
  - Same as inputDate1, but the range of years should be customizable by page author
- **Code summary**
  - Java class
    - Move getYears, getMonths, getDays out of separate DateChoices class, and into component class.
    - Add startYear and endYear bean properties
      - setStartYear and setEndYear change what will be returned by getYears
  - Composite component
    - Interface: add startYear and endYear attributes
    - Implementation, use c:set to call setStartYear/setEndYear

Java Class (Beginning)

```
@FacesComponent("coreservlets.DateComponent2")
public class DateComponent2 extends DateComponent1 {
  private int[] days;
  private int[] years;
  private int startYear = 1900;
  private int endYear = 2100;
  private static Map<String, Integer> months;

  static {  // Months never change, but years and days do
    months = new LinkedHashMap<String, Integer>();
    String[] names = new DateFormatSymbols().getMonths();
    for (int i = 0; i < 12; i++) {
      months.put(names[i], i + 1);
    }
  }
  public DateComponent2() {
    days = DateUtils.intArray(1, 31);
    years = DateUtils.intArray(startYear, endYear);
  }
```

Basic backing component methods (encodeBegin, getConvertedValue, etc.) inherited unchanged from parent class that was shown in previous example.

Java Class (Continued)

```
  public int getStartYear() {
    return(startYear);
  }

  public void setStartYear(int startYear) {
    this.startYear = startYear;
    years = DateUtils.intArray(startYear, endYear);
  }

  public int getEndYear() {
    return(endYear);
  }

  public void setEndYear(int endYear) {
    this.endYear = endYear;
    years = DateUtils.intArray(startYear, endYear);
  }
```


Java Class (Continued)

```
  public int[] getDays() {
    return(days);
  }

  public int[] getYears() {
    return(years);
  }

  public Map<String, Integer> getMonths() {
    return(months);
  }
}
```

Component File: Outline

```
<!DOCTYPE html ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:cc="http://xmlns.jcp.org/jsf/composite"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:c="http://xmlns.jcp.org/jsp/jstl/core">

<cc:interface componentType="...">
  ...
</cc:interface>
<cc:implementation>
  ...
</cc:implementation>

</html>
```


Component File: Interface Section

```
<cc:interface componentType="coreservlets.DateComponent2">
  <cc:attribute name="value" type="java.util.Date"/>
  <cc:attribute name="startYear" type="java.lang.Integer"
                default="2012"/>
  <cc:attribute name="endYear" type="java.lang.Integer"
                default="2040"/>
</cc:interface>
```

Component File: Implementation Section

```
<cc:implementation>
  <c:set target="#{cc}" property="startYear"
         value="#{cc.attrs.startYear}"/>
  <c:set target="#{cc}" property="endYear"
         value="#{cc.attrs.endYear}"/>
  <h:selectOneMenu id="day" converter="javax.faces.Integer">
    <f:selectItems value="#{cc.days}"/>
  </h:selectOneMenu>
  <h:selectOneMenu id="month" converter="javax.faces.Integer">
    <f:selectItems value="#{cc.months}"/>
  </h:selectOneMenu>
  <h:selectOneMenu id="year" converter="javax.faces.Integer">
    <f:selectItems value="#{cc.years}"/>
  </h:selectOneMenu>
</cc:implementation>
```

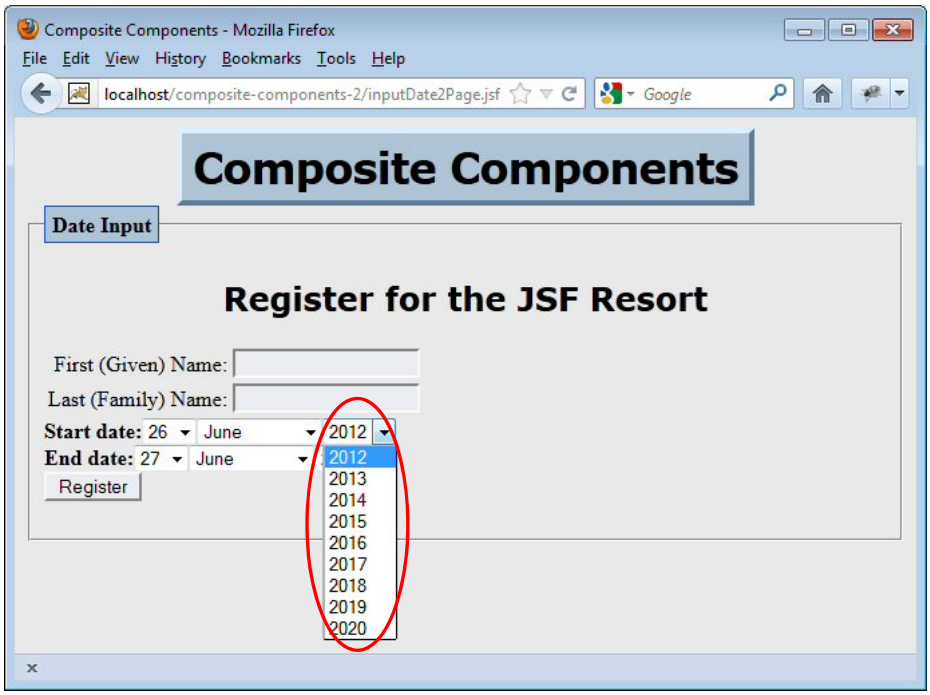

Example Usage: Reservations at the JSF Resort

- **Idea**
  - Same as previous: let user make a hotel reservation.
    - But, range of years is customizable by page author
- **Managed Bean**
  - Unchanged from previous example
- **Page that uses component**
  - Collects name, checkin (arrival) date, and checkout (departure) date
    - Gives custom ranges of years
- **Results page**
  - Unchanged from previous example

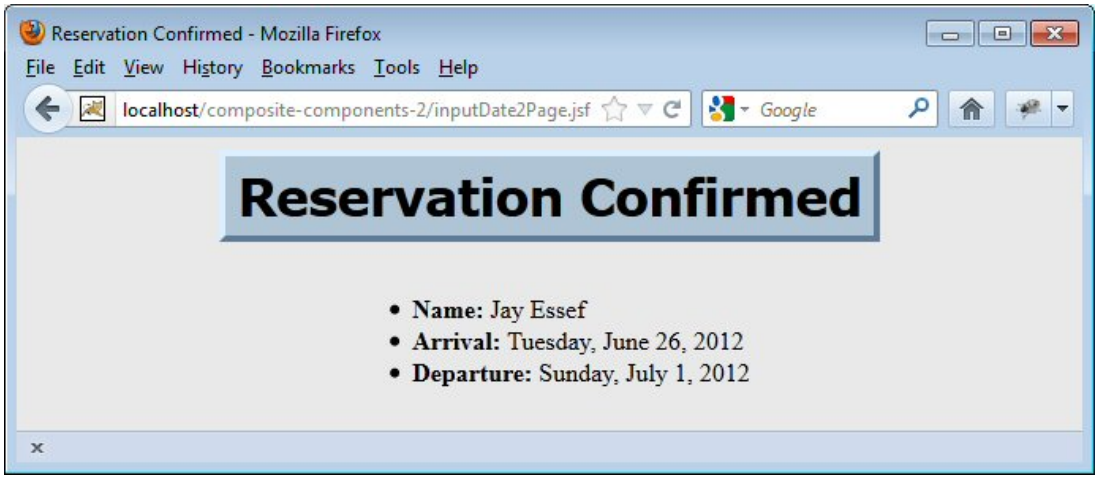
Facelets Page that Uses Component

```
...
<h:form id="registrationForm">
   <h2>Register for the JSF Resort</h2>
   <utils:inputName2 value="#{resortBean}"
                     firstNamePrompt="First (Given) Name"
                     lastNamePrompt="Last (Family) Name"/>
   <b>Start date:</b>
   <utils:inputDate2 value="#{resortBean.startDate}"
                     startYear="2012" endYear="2020"/><br/>
   <b>End date:</b>
   <utils:inputDate2 value="#{resortBean.endDate}"
                     id="checkoutDate"
                     startYear="2012" endYear="2021"/>
   <h:message for="registrationForm:checkoutDate"
              styleClass="error"/><br/>
   <h:commandButton action="#{resortBean.register}"
                    value="Register"/><p/>
</h:form>
...
```


Result: Page that Uses Component



Result: Results Page



Using Ajax in Composite Components

Ajaxified Components: Idea

- **No Ajax**
  - Once page loaded, the menus always look the same
    - Changes to one menu do not affect others.
    - So, in our example, the choice for days is always 1-31, even for months that should have less than 31 days
      - In the code, Feb 30 would wrap around and become March 2
- **Ajax**
  - Changes to one menu affect how others look
    - Changing the month affects the number of days
    - Changing the year might affect the number of days in February (depending on leap years)

Ajaxified Components: Code Summary

- **Java code**
  - Same as in previous examples
- **Composite component**

```
…
<cc:implementation>
    <h:selectOneMenu id="month"
                              converter="javax.faces.Integer">
        <f:selectItems value="#{cc.months}"/>
        <f:ajax render="day" execute="@all"/>
    </h:selectOneMenu>
    …
</cc:implementation>
```

Using Ajax in Composite Components

- **render attribute**
  - Give the id exactly as listed in the component file
    - No need to put form id and client id in front, as we will have to do when wrapping up JavaScript components
    - JSF will automatically resolve the id properly
- **execute attribute**
  - Use @all
  - JSF will automatically treat @all as only the elements inside the component, not all elements in the page

Java Class (First Half)

```
@FacesComponent("coreservlets.DateComponent3")
public class DateComponent3 extends DateComponent2 {
  private static final int[] DAYS_28 =
    DateUtils.intArray(1, 28);
  private static final int[] DAYS_29 =
    DateUtils.intArray(1, 29);
  private static final int[] DAYS_30 =
    DateUtils.intArray(1, 30);
  private static final int[] DAYS_31 =
    DateUtils.intArray(1, 31);
```

Java Class (Second Half)

```
@Override
public int[] getDays() {
  Date date = (Date)getValue();
  Calendar cal = new GregorianCalendar();
  cal.setTime(date);
  int month = cal.get(Calendar.MONTH) + 1;
  int year = cal.get(Calendar.YEAR);
  int daysInMonth =
    DateUtils.daysInMonth(month, year);
  switch (daysInMonth) {
    case 28: return(DAYS_28);
    case 29: return(DAYS_29);
    case 30: return(DAYS_30);
    default: return(DAYS_31);
  }
}
```


Helper Class

```
public class DateUtils {
  public static int daysInMonth(int month, int year) {
    if (month == 2) {
      if (isLeapYear(year)) {
        return (29);
      } else {
        return (28);
      }
    } else if (month == 4 || month == 6 || month == 9 || month == 11) {
      return (30);
    } else {
      return (31);
    }
  }

  private static boolean isLeapYear(int year) {
    return((year % 4 == 0) &&
           ((year % 400 == 0) || (year % 100 != 0)));
  }

  // Several other Date-related utilities shown elsewhere
}
```

Component File: Interface Section

```
<cc:interface componentType="coreservlets.DateComponent3">
  <cc:attribute name="value" type="java.util.Date"/>
  <cc:attribute name="startYear" type="java.lang.Integer"
                default="2012"/>
  <cc:attribute name="endYear" type="java.lang.Integer"
                default="2040"/>
</cc:interface>
```


Component File: Implementation Section

```
<cc:implementation>
  <c:set target="#{cc}" property="startYear"
         value="#{cc.attrs.startYear}"/>
  <c:set target="#{cc}" property="endYear"
         value="#{cc.attrs.endYear}"/>
  <h:selectOneMenu id="day" converter="javax.faces.Integer">
    <f:selectItems value="#{cc.days}"/>
  </h:selectOneMenu>
  <h:selectOneMenu id="month" converter="javax.faces.Integer">
    <f:selectItems value="#{cc.months}"/>
    <f:ajax render="day" execute="@all"/>
  </h:selectOneMenu>
  <h:selectOneMenu id="year" converter="javax.faces.Integer">
    <f:selectItems value="#{cc.years}"/>
    <f:ajax render="day" execute="@all"/>
  </h:selectOneMenu>
</cc:implementation>
```

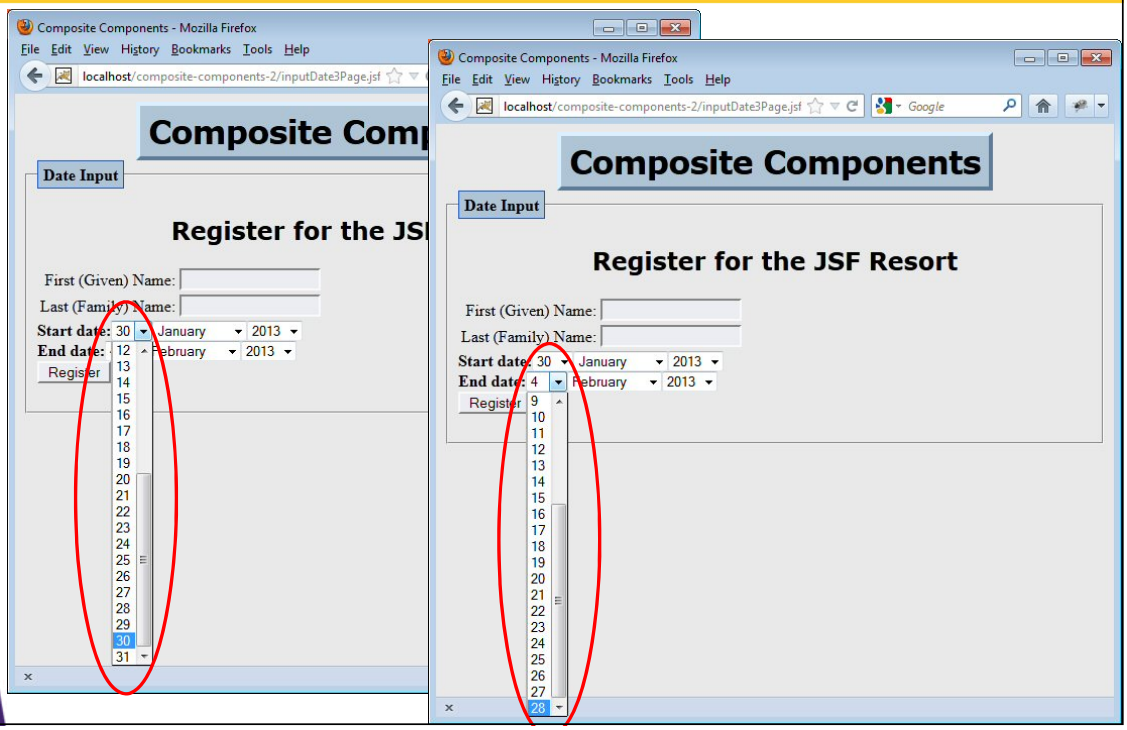
Example Usage: Reservations at the JSF Resort

- **Idea**
  - Same as previous: let user make a hotel reservation.
    - But, choices for days should match the month (and year, in the case of leap years and February)
- **Managed Bean**
  - Unchanged from previous example
- **Page that uses component**
  - Same as last example: collects name, checkin date, and checkout date. Gives custom ranges of years.
- **Results page**
  - Unchanged from previous example

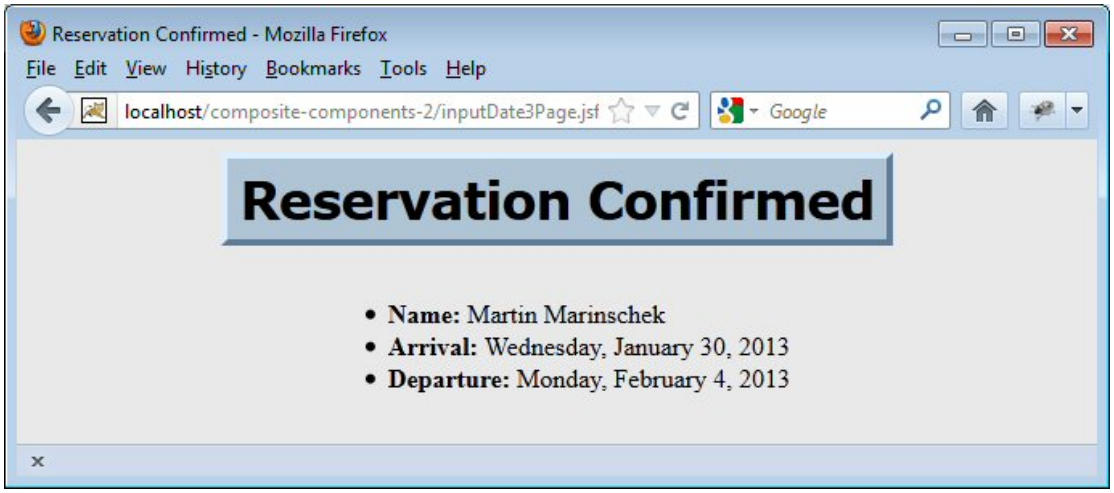
Facelets Page that Uses Component

```
...
<h:form id="registrationForm">
   <h2>Register for the JSF Resort</h2>
   <utils:inputName2 value="#{resortBean}"
                     firstNamePrompt="First (Given) Name"
                     lastNamePrompt="Last (Family) Name"/>
   <b>Start date:</b>
   <utils:inputDate3 value="#{resortBean.startDate}"
                     startYear="2012" endYear="2020"/><br/>
   <b>End date:</b>
   <utils:inputDate3 value="#{resortBean.endDate}"
                     id="checkoutDate"
                     startYear="2012" endYear="2021"/>
   <h:message for="registrationForm:checkoutDate"
              styleClass="error"/><br/>
   <h:commandButton action="#{resortBean.register}"
                    value="Register"/><p/>
</h:form>
...
```

Result: Page that Uses Component



Result: Results Page



Wrap-Up

Summary

- **Assemble final value out of pieces by extending UIInput and overriding methods**
  - getFamily
    - return("javax.faces.NamingContainer");
  - encodeBegin
    - find components with findComponent("local-id"), set vals
  - getSubmittedValue
    - return(this);
  - getConvertedValue
    - findComponent, getSubmittedValue, convert and combine
- **Call setter methods based on attribute values**
  - Most portable approach: use c:set
- **Embed Ajax behavior**
  - Use execute="@all"

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