BAB 10

Advanced JSF

10.1 PENDAHULUAN

Pada pelajaran sebelumnya, kita pelajari tentang framework Java Server Faces, bagaimana JSF menggabungkan front controller servlet, penanganan action, dan sekumpulan komponen UI untuk mempermudah pengembangan sebuah aplikasi dalam arsitektur Model – View – Controller. Kita telah melihat bagaimana JSF menghasilkan pemrograman Swing dalam lingkungan web, dengan arsitektur komponen berbasis event.

Dalam pelajaran ini, kita akan mendiskusikan object FacesContext, dimana object ini menangani semua informasi kontekstual dalam framework JSF. Kita akan melihat kumpulan komponen lain yang tersedia yang dapat digunakan oleh framework : converter validator dan tipe. Kita juga akan mempelajari bagaimana membuat komponen kita sendiri, dan bagaimana mengembangkan kumpulan tag-tag kita sendiri sehingga komponen yang kita buat dapat lebih mudah direpresentasikan dalam halaman JSP.

10.2 FacesContext

Dalam diskusi sebelumnya, secara ringkas telah kita lihat bagaimana membuat penggunaan object FacesContext untuk mengakses Map dari object yang sedang dalam session user. Kita akan mencakupi topik tersebut dan penggunaan object FacesContext lainnya secara terperinci di sini.

Sebagai pokok tinjauan, instance dari object FacesContext dapat diperoleh dengan memanggil method FacesContext.getCurrentInstance().

10.2.1 FacesContext dan Component Tree

Untuk setiap view pembuatan elemen UI JSF, terdapat kesesuaian component tree pada model tersebut. Spesifikasi JSF membutuhkan semua pemakaian JSF yang menyimpan suatu component tree dalam object FacesContext. JSF mengizinkan pengembang mengakses semua komponen user interface dan datanya.

Terdapat beberapa hal yang dapat kita lakukan dalam mengakses component tree jenis ini. Kita dapat menambahkan komponen pada view secara terencana, mengubah atau menambah converter atau validator secara dinamis, atau menghilangkan komponen.

Walaupun sangat sering, yang kita lakukan adalah menggunakan kemampuan ini untuk menyediakan tampilan yang mengarah langsung dalam framework Faces. Ini dapat dilakukan dengan merubah component tree yang ada untuk diakses oleh user dengan component tree yang mengubah halaman lain.

Di bawah ini adalah contoh potongan kode yang mengerjakan tugas ini, dengan baris yang dicetak tebal :

```
...
    String targetPage = "/newPage.jsp";
    FacesContext context = FacesContext.getCurrentInstance();
    context.getApplication().getViewHandler().createView(context, targetPage);
...
```

Method createView membuat salinan dari component tree yang membuat targetPage dan menempatkannya sebagai view component tree. Setelah framework berjalan menampilkan isinya lagi, framework akan menampilkan isi dari halaman sasaran yang dituju.

10.2.2 FacesContext dan External Context

Object ExternalContext yang dapat diperoleh lewat FacesContext memberi kita akses ke lingkungan framework yang sedang berjalan. Dalam kasus kita (aplikasi web), object tersebut memberi kita akses ke object HttpServletRequest yang menggambarkan request yang ada, Map dari object yang disimpan dalam session user, sebaik object ServletContext yang menggambarkan konteks dari seluruh aplikasi web.

Untuk menerima object HttpServletRequest :

```
    FacesContext context = FacesContext.getCurrentInstance();
    HttpServletRequest request = (HttpServletRequest)context.getExternalContext()
       .getRequest();
```

Sayangnya, tidak ada cara untuk memiliki akses secara langsung ke object HttpSession yang diasosiasikan dengan session kita, tetapi object disimpan di dalamnya sehingga dapat diakses melewati object Map :

```
    Map sessionMap = context.getExternalContext().getSessionMap();
```

Penerimaan ServletContext menggambarkan seluruh aplikasi kita yang dilakukan oleh kode berikut :

```
    ServletContext servletContext(ServletContext)context.getExternalContext().getContext();
```

Suatu waktu kita telah mengakses object ini, kemudian kita dapat membuat teknik pemrograman web lainnya.

10.3 Validator

Dalam pelajaran kedua kita pada framework Struts, kita telah mendiskusikan bagaimana pentingnya validasi data dalam berbagai aplikasi berbasis data. Seperti perkataan orang : garbage in- garbage -out; beberapa hasil aplikasi selalu dikendalikan oleh ketepatan data yang menangani.

JSF juga telah mengenal kebutuhan ini untuk validasi dan menyediakan kita dengan kumpulan validator yang kita dapat gunakan keluar dari aplikasi kita. Juga dapat memberi kita beberapa cara untuk memberi kode validasi ke dalam komponen input data kita.

10.3.1 Validator standar JSF

JSF menyediakan 3 validator standar :

- DoubleRangeValidator – memeriksa jika nilai dalam field masukan dapat ditukar ke dalam double dengan memberikan nilai minimun dan maksimum. **Tag JSP :** validateDoubleRange. **Atribut :** minimum, maximum
- LengthValidator – memeriksa jika nilai dalam field masukan dapay ditukar ke dalam String dengan panjangnya disediakan nilai minimum dan maksimum. **Tag JSP :** validateLength. **Atribut :** minimum, maximum.
- LongRangeValidator – memeriksa jika nilai dalam field masukan dapat ditukar ke dalam Long dan disediakan nilai minimum dan maksimum. **Tag JSP :** validateLongRange. **Atribut :** minimum, maximum.

Semua tag-tag JSP untuk validator dapat ditemukan di dalam library tag bagian utama. Sebagai pengingat, untuk menggunakan tag-tag dalam library utama, kita tambahkan baris berikut pada atas halaman :

```
<%@taglib uri ="http://java.sun.com/jsf/core/" prefix="f" %>
```

10.3.2 Menggunakan validator standar

Penggunaa validator standar semudah menyisipkan tag JSP untuk validator dalam tubuh komponen masukan untuk divalidasi. Lihat contoh berikut :

```
...
      <h:outputLabel for="password">
          <h:outputText value="Password : "/>
      </h:outputLabel>
      <h:inputSecret id="password" value="#{loginPage.password}">
         <f:validateLength minimum=”4”/>
      </h:inputSecret>
<br/>

      <h:commandButton action="#{loginPage.performLogin}" value="Login"/>
   </h:form>
...
```

Kode ini adalah kode JSP untuk halaman login yang kita pakai dalam pelajaran sebelumnya menggunakan JSF, dengan perubahan seperti itu field password hanya akan menerima sebuah masukan dengan panjang minimum adalah 4. Rincian perubahan dilakukan adalah :

- tag mendefinisikan field password (<h:inputSecret>) yang diubah hingga tag tidak lama ditutup sendiri pada baris yang sama dengan yang dideklarasikan sebelumnya. Penggunaan tag </h:inputSecret) untuk menutup dirinya sendiri.
- tag mendefinisikan validasi untuk dilakukan (<f:validateLength>) yang disisipkan antara pembuka dan penutup tag <h:inputSecret>.

Jika kita ingin menerima masukan dengan panjang **maksimum** dan bukan minimum, kita dapat mengganti atribut minimum dengan atribut maximum, dan menetapkan panjang max. Kita juga dapat menggunakan kedua atribut pada waktu yang sama, untuk mendefinisikan jarak antara masukan yang akan diterima. Atribut ini bekerja dengan cara yang sama untuk tag-tag <f:validateLongRange> dan <f:validateDoubleRange>, hanya perbedaan tipe data yang akan diterima.

10.3.3 Customized Validation

Validator standar dibatasi pada apa yang mereka mampu lakukan; sangat mungkin kita perlu membuat kode validasi untuk memeriksa ketepatan data pada aplikasi kita. Terdapat tiga cara untuk melakukannya : kita dapat a) memperluas class dari komponen UI yang menerima masukan kita sehingga kita dapat meng-override method validasinya; b) membuat method validasi eksternal; atau c) membuat pemakaian Validator kita secara terpisah, meregisternya dalam framework, kemudian di-plug ke dalam komponen UI.

Pemasalahan dengan method pertama, adalah memperluas class untuk di-override method validasinya, yang berupa non-portable : validasi hanya dapat ditampilkan ketika menggunakan UIComponent yang khusus. Dalam pelajaran ini, kita akan lebih memusatkan pada 2 pilihan yang lain.

10.3.3.1 Menggunakan method validasi eksternal

Pilihan pertama yang akan kita diskusikan dalam pembuatan validasi kita sendiri adalah membuat method validasi eksternal. Prosedur ini mirip dengan pembuatan method aplikasi yang akan menangani event action : kita juga perlu membuat method mengikuti kumpulan peraturan di dalam JavaBean yang diatur oleh framework dan mengikat method nya ke komponen UI yang tepat.

10.3.3.2 Membuat method validasi eksternal Signature method

Method yang kita buat harus memenuhi aturan-aturan berikut :

- Method harus dideklarasikan secara public, dengan return type void.
- Tidak ada batasan pada penamaan method.
- Method harus membawa parameter dengan urutan berikut : FacesContext ctxt, UIInput component, Object value
- Method harus dideklarasikan untuk memberi ValidatorException

Jika kita membuat method validasi sendiri untuk field password dalam contoh sebelumnya, salah satunya harus hanya menerima huruf dan angka pengganti, signature method ini akan tampak seperti berikut :

```
public void validatePassword(FacesContext ctxt, UIInput component, Object value)
    throws ValidatorException
```

10.3.3.3 Pemakaian Method

Dalam pemakaian method kita, kita dapat menggunakan data yang disediakan lewat parameter. Object FacesContext menyediakan kita mengakses sumber eksternal seperti object dalam jangkauan request dan session, sebaik object lain dalam framework Faces. Object UIInput adalah instance dari komponen masukan yang memerlukan validasi : mempunyai instance dari object ini memberikan kita akses pada status komponen. Akhirnya, nilai object adalah nilai dalam komponen yang memerlukan validasi.

Proses validasinya sederhana : jika framework tidak menerima beberapa ValidatorException, masukan diterima. Jika framework menerima ValidatorException, selanjutnya proses dihentikan, dan halaman yang berisi komponen masukan akan ditampilkan kembali.

Di bawah ini adalah contoh dari implementasi untuk method kita yang akan tampak seperti :

```
public void validatePassword(FacesContext ctxt, UIInput component, Object value)
       throws ValidatorException {

    if (null == value)
        return;

    if ( ! (isPasswordValid(value.toString() ) ) {
        FacesMessage message = new FacesMessage(“Input error”, “Password is not valid”);
        throw new ValidatorException(message);
    }
}

protected boolean isPasswordValid(String password) {
...
```

Kita temukan class baru dalam contoh di atas : FacesMessage. Class ini secara sederhana memodelkan message dalam framework Faces. Parameter pertama yang dilewatkan dalam constructor-nya adalah summary, kedua rincian dari message yang ada. Ini digunakan untuk mengindikasikan ke framework message mana yang akan dikenal oleh framework sebagai pesan error.

Untuk dapat benar-benar menampilkan pesan error yang sudah di-generate, pengembang harus menambahkan tag <h:messages> atau <h:message> dalah halaman JSP. Akan ditunjukkan sebagai berikut :

```
...
          <h:outputLabel for="password">
                 <h:outputText value="Password : "/>
          </h:outputLabel>
          <h:inputSecret id="password" value="#{loginPage.password}">
               <f:validateLength minimum=”4”/>
          </h:inputSecret>
            <h:message for=”password” styleClass=”error”/>
<br/>

          <h:commandButton action="#{loginPage.performLogin}" value="Login"/>
     </h:form>
...
```

Tag <h:message> hanya menampilkan message untuk komponen yang khusus, ditandai oleh nilai dalam atribut for. Kita juga menentukan class style CSS yang akan dipakai pada message. Tag <h:messages> menampilkan semua message yang cocok untuk semua komponen dalam halaman.

10.3.4 Membuat implementasi Validator yang terpisah

Daripada kita membuat method dalam aturan JavaBean, kita juga dapat membuat class yang terpisah yang mengimplementasikan interface Validator. Interface ini mendefinisikan satu method : method validasi yang signature nya tepat dari sebuah method validasi eksternal : bersifat public, kembaliannya kosong, dan dilewati instance dari FacesContext, UIInput, dan class-class Object sebagai parameter.

Dalam peraturan pemakaian method validasi, tidak ada perbedaan diantara pemakaian Validator yang terpisah dari method validasi eksternal. Perbedaan apa yang ada pada keduanya adalah bagaimana mereka digunakan : sebuah method validasi eksternal lebih digunakan untuk kode validasi yang khusus ke komponen yang particular, sementara pemakaian Validator yang terpisah digunakan untuk memuat kode validasi dengan tujuan umum yang akan digunakan kembali secara luas dalam aplikasi Anda.

10.3.4.1 Register komponen sebuah Validator

Untuk menggunakan komponen Validator kita sendiri, pertama kita harus me-register nya sehingga dapat dikenal oleh framework JSF. Ini dapat dilakukan dengan menempatkan catatan konfigurasi yang dimasukkan dalam file faces-config.xml.

Dapat dilihat contoh di bawah ini :

```
<validator>
    <description>A validator for checking the password field</description>
    <validator-id>AlternatingValidator</validator-id>
    <validator-class>jedi.sample.jsf.validator.AlternatingValidator</validator-class>
</validator>
```

Seperti yang dapat kita lihat, catatan konfigurasi untuk validator baru semudah mendefinisikan id dimana akan diarahkan oleh framework, sebaik nama class yang benar-benar memenuhi persyaratan yang mengimplementasikan kemampuan validasi.

10.3.4.2 Menggunakan komponen validator

Untuk menggunakan komponen validator kita hanya me-registernya, kita gunakan tag <f:validator> dan sediakan id validator dalam atribut validatorId nya. Berikut gambarannya :

```
...
          <h:outputLabel for="password">
                <h:outputText value="Password : "/>
          </h:outputLabel>
          <h:inputSecret id="password" value="#{loginPage.password}">
               <f:validator validatorId=”AlternatingValidator”/>
          </h:inputSecret>
            <h:message for=”password” styleClass=”error”/> <br/>

          <h:commandButton action="#{loginPage.performLogin}" value="Login"/>
     </h:form>
...
```

10.3.4.3 Menambahkan atribut pada validator kita

Jika kita lihat pada validator standar validateLength yang disediakan oleh framework JSF, kita dapat lihat validator tersebut berisi dua atribut dimana operasi dapat dilakukan lebih lanjut. Demikian pula dapat kita miliki atribut untuk validator kita sendiri.

Untuk melakukannya, kita tambahkan properti dalam class kita untuk setiap atribut yang akan mendukung dan mengimplementasikan method get dan set masing-masing. Selanjutnya, kita masukkan entry setiap atribut dalam catatan konfigurasi kita untuk validator.

Untuk melakukannya, kita tambahkan properti dalam class kita untuk setiap atribut yang akan mendukung dan mengimplementasikan method get dan set masing-masing. Selanjutnya, kita masukkan entry setiap atribut dalam catatan konfigurasi kita untuk validator.

```
package jedi.sample.jsf.validator;

public class AlternatingValidator implements Validator {
    private Integer interval;

    public Integer getInterval() {
        return interval;
    }

    public void setInterval(Integer interval) {
        this.interval = interval;
    }

    public void validate(FacesContext ctxt, UIInput component, Object value)
        throws ValidatorException {

        if (null == value || interval == null)
            return;

        if ( ! (isPasswordValid(value.toString() ) ) {
            FacesMessage message = new FacesMessage(“Input error”, “Password is not valid”);
            throw new ValidatorException(message);
        }
    }

    protected boolean isPasswordValid(String password) {
   ...
```

```
<validator>
    <description>A validator for checking the password field</description>
    <validator-id>AlternatingValidator</validator-id>
    <validator-class>jedi.sample.jsf.validator.AlternatingValidator</validator-class>
    <attribute>
         <attribute-name>interval</attribute-name>
         <attribute-class>java.lang.Integer</attribute-class>
    </attribute>
</validator>
```

Nilai yang dapat disediakan untuk atribut ini dengan menggunakan tag <f:attribute> :

```
...
          <h:outputLabel for="password">
                 <h:outputText value="Password : "/>
          </h:outputLabel>
          <h:inputSecret id="password" value="#{loginPage.password}">
               <f:validator validatorId=”AlternatingValidator”/>
              <f:attribute name=”interval” value=”5”/>
          </h:inputSecret>
            <h:message for=”password” styleClass=”error”/> <br/>

          <h:commandButton action="#{loginPage.performLogin}" value="Login"/>
     </h:form>
...
```

10.4 Converter

Converter adalah komponen set penting lainnya yang disediakan JSF. Converter menyediakan solusi untuk permasalahan dasar yang mengganggu programer web : bagaimana merubah nilai yang diberikan oleh user dari string yang direpresentasikan dalam format atau tipe yang tepat yang digunakan secara internal oleh server. Converter juga bersifat dua arah : converter juga dapat digunakan untuk merubah bagaimana data internal ditampilkan ke user.

Converter mampu melakukan hal ini dengan mendefinisikan method getAsString() dan getAsObject() yang dapat dipanggil oleh framework pada saat yang tepat. Method getAsString dipanggil untuk memberikan representasi String ke data, sementara getAsObject digunakan untuk merubah String ke dalam tipe yang diminta.

Converter dihubungkan ke tipe masukan dengan menggunakan atribut converter yang built-in ke beberapa komponen masukan yang disediakan oleh Faces. Berikut ilustrasi contohnya, dimana mengubah tag inputText sehingga tentu saja akan memenuhi perubahan Integer :

```
<h:inputText converter=”Integer” value=”#{myFormBean.myIntProperty}”/>
```

Selain dari komponen inputText, komponen inputSecret, inputHidden, dan outputText juga dapat merubah atribut.

Integer di sini adalah converter ID yang diberikan pada salah satu dari elemen converter standar yang disediakan oleh framework. Berikut converter standar lainnya :

Converter Class	***Converter ID***
BigDecimalConverter	BigDecimal
BigIntegerConverter	BigInteger
IntegerConverter	Integer
ShortConverter	Short
ByteConverter	Byte
ShortConverter	Short
CharacterConverter	Character
FloatConverter	Float
DoubleConverter	Double
BooleanConverter	Boolean
DateTimeConverter	DateTime
NumberConverter	Number

Diantara kumpulan converter ini, DateTimeConverter dan NumberConverter memiliki arti khusus. Kumpulan converter lainnya merupakan non-configurable; hanya dapat digunakan seperti pada contoh-contoh di atas. DateTimeConverter dan NumberConverter, bagaimanapun, menggunakan tag-tag khusus. Lewat tag ini mereka mengekspos atribut dimana developer dapat mengatur sifatnya sendiri.

10.4.1 DateTimeConverter

DateTimeConverter dapat digunakan untuk mengubah masukan ke dalam instance dari java.util.Date. DateTimeConverter menyediakan banyak atribut yang mana developer dapat menetapkan format yang digunakan dalam konversi. Sangat penting dicatat, bahwa format yang ditetapkan *dijalankan* ketika user memberi masukannya. Jika user tidak menetapkan masukannya dalam format yang telah ditetapkan dalam atribut, akan terjadi kesalahan pengkonversian, dan framework akan berhenti memproses data selanjutnya.

Berikut ini atribut yang tersedia :

- dateStyle – satu dari style date yang didefinisikan dalam java.text.DateFormat. Nilai yang memungkinkan yaitu : default, short, long, medium, atau full.
- parseLocale - lokasi yang digunakan selama pengkonversian.
- pattern – pola format yang digunakan dalam pengkonversian. Jika nilai untuk atribut ini ditetapkan, sistem akan mengabaikan beberapa nilai untuk dateStyle, timeStyle, dan type.
- timeStyle – satu dari style time yang didefinisikan dalam java.text.DateFormat. Nilai yang mungkin untuk atribut ini yaitu : default, short, medium, long, atau full.
- timeZone – zona waktu yang digunakan.
- type - String yang mendefinisikan apakah masukan akan berupa instance tanggal atau waktu, atau keduanya. Nilai yang mungkin yaitu : date, time, dan both. Nilai defaultnya adalah date.

Atribut ini disediakan untuk developer dengan tag <f:convertDateTime>.

Mari kita buat contoh singkat dalam penggunaan dari DateTimeConverter. Pertama-tama, buat aplikasi web kosong yang telah siap untuk JSF, seperti setiap perintah pada pelajaran sebelumnya. Kemudian, mari memulainya dengan membuat JavaBeans sederhana yang akan menangani masukan date nantinya :

```
import java.util.Date;

public class DateBean {
    private Date date;

    public Date getDate() {
        return date;
    }

    public void setDate(Date date) {
        this.date = date;
    }
}
```

Kemudian, tambahkan entri konfigurasi dalam faces-config.xml :

```
    <managed-bean>
        <description>
            This bean serves as the backing model for our date conversion example
        </description>
        <managed-bean-name>dateBean</managed-bean-name>
        <managed-bean-class>jedi.sample.DateBean</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
         <managed-property>
             <property-name>date</property-name>
             <null-value/>
         </managed-property>
    </managed-bean>
```

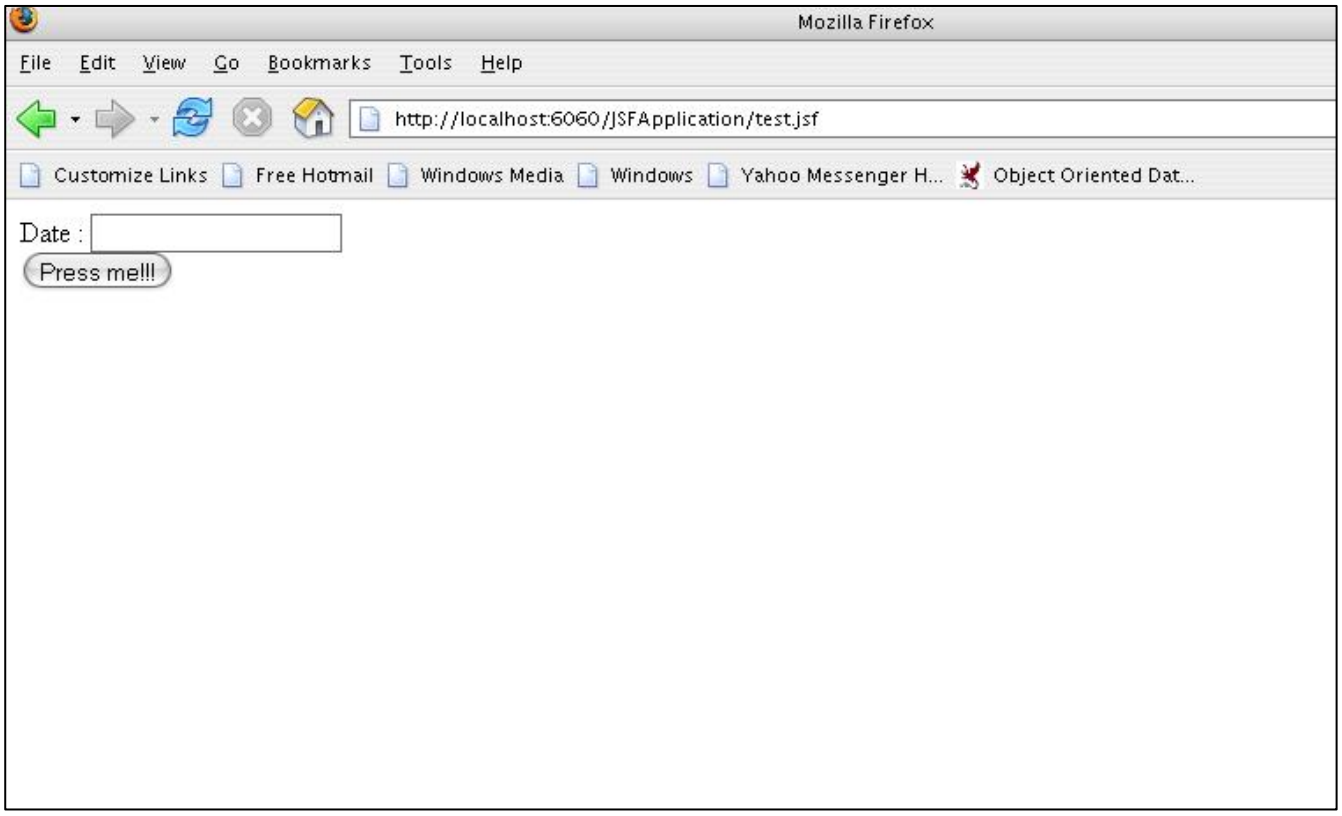
Akhirnya, kita buat file JSP yang akan men-generate view nya :

```
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>

<f:view>
   <h:form id="testForm">

      <h:outputLabel for="dateField">Date : </h:outputLabel>

      <h:inputText id="dateField" value="#{dateBean.date}" required="true">
        <f:convertDateTime pattern="M/d/yy"/>
      </h:inputText> <br/>

      <h:message for="dateField"/> <br/>

      <h:commandButton id="button" value="Start conversion"/>
   </h:form>
</f:view>
```

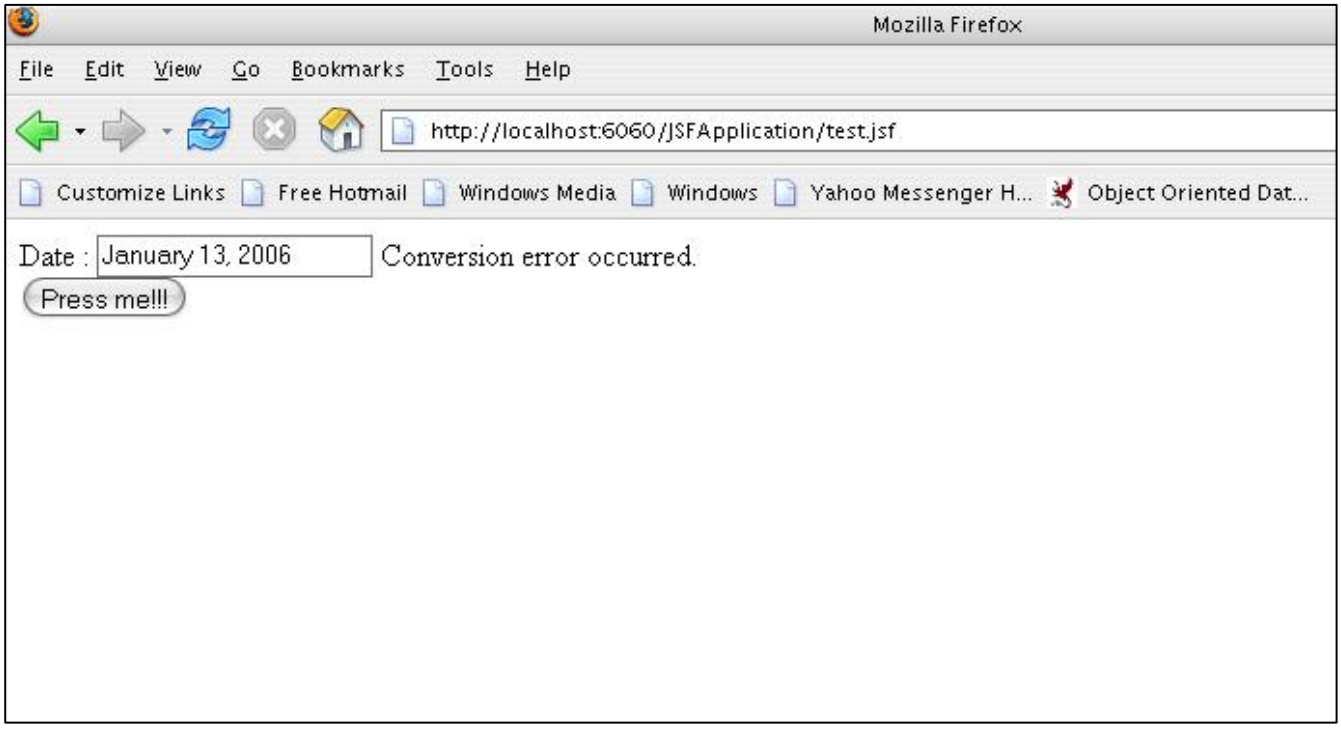
Beberapa catatan untuk contoh halaman JSP :

- Untuk DateTimeConverter, telah ditetapkan pola M/d/yy. Karena converter menjalankan pola ini pada user, user akan memiliki masukan seperti 1/13/06 sebagai tanggal sebelum dia dapat melanjutkan ke masukan lain. Untuk pola yang tepat, periksalah dokumentasi API untuk java.text.SimpleDateFormat.
- Kita telah tambahkan atribut yang diperlukan dengan nilai yang benar/true untuk field masukan. Hal ini untuk memastikan bahwa user tidak memasukkan masukan kosong.
- Kita telah menggunakan tag <h:message> yang dihubungkan dengan masukan dateField kita. Hal ini untuk memastikan bahwa beberapa message di-generate oleh field masukan(seperti pada generate oleh kesalahan konversi) yang ditampilkan.
- Atribut action untuk commandButton yang dengan sengaja dibuat salah, sehingga kita dapat memfokuskan pada message yang di-generate oleh converter.

Loading aplikasi web ke dalam server yang cocok (dalam kasus kita dimisalkan dari AppServer 8.1) dan mengakses halaman yang telah dibuat akan menghasilkan tampilan berikut ini :



Cobalah memasukkan masukan January 13, 2006 akan dihasilkan seperti berikut :



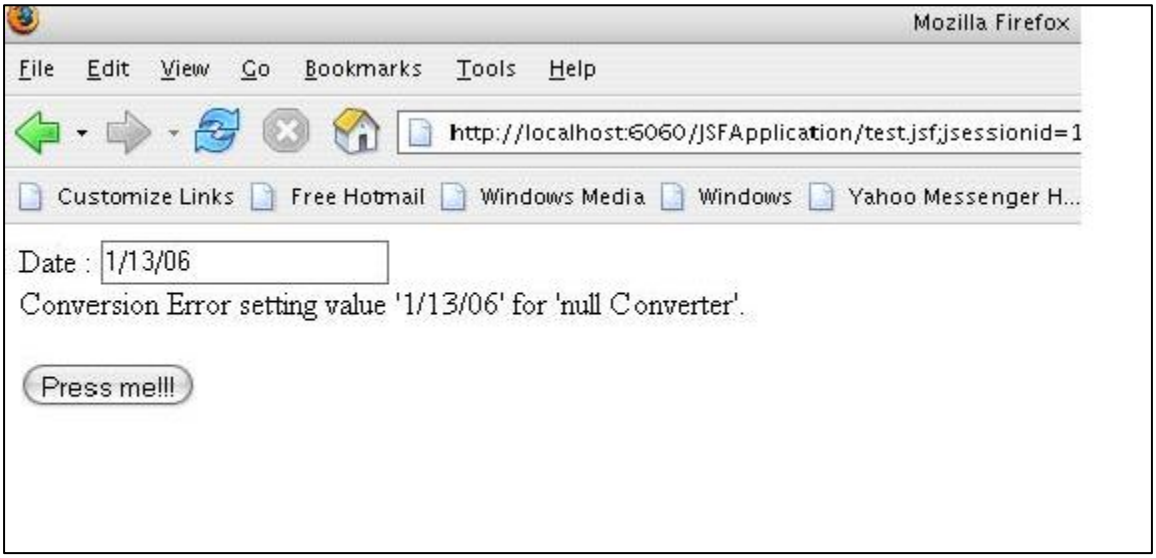
Memasukkan tanggal dalam format yang cocok tidak akan menghasilkan perubahan pada halaman kita. Ini mengindikasikan bahwa data telah sukses diubah dan disimpan dalam bean kita.

Apa yang akan terjadi jika kita tidak menggunakan converter?

Untuk menggambarkan keuntungan dari converter, mari mengubah halaman JSP seperti menghilangkan converter :

```
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>

<f:view>
  <h:form id="testForm">

    <h:outputLabel for="dateField">Date : </h:outputLabel>

    <h:inputText id="dateField" value="#{dateBean.date}" required="true"/>
    <br/>

    <h:message for="dateField"/> <br/>

    <h:commandButton id="button" value="Start conversion"/>
  </h:form>
</f:view>
```

Sekarang, memasukkan tanggal dengan format apapun akan menyebabkan kesalahan berikut :



Hal ini dikarenakan properti yang diasosiasikan dengan field masukan kita adalah object Date. Framework tidak dapat secara otomatis merubah representasi String dari masukan ke dalam

tanggal, sehingga akan mencari object Converter yang akan menanganinya, karena tidak ada sehingga menyebabkan kesalahan.

10.4.2 NumberConverter

Converter khusus lainnya yang akan kita lihat yaitu NumberConverter. Converter ini digunakan untuk mengubah masukan angka atau menjalankan format khusus padanya. Berikut atribut dimana kita dapat mengontrol tingkah laku dari converter :

- currencyCode – sebuah kode mata uang ISO 4217 yang digunakan ketika formatnya mata uang
- currencySymbol – simbol mata uang yang digunakan dalam format mata uang
- groupingUsed – mengindikasikan apakah simbol pengelompokkan akan digunakan atau tidak (contohnya, komma setiap tiga digit).
- integerOnly – mengindikasikan apakah bagian integer hanya dari angka akan ditampilkan atau tidak. Untuk masukan, ini berarti bagian desimal akan dipotong sebelum data disimpan ke dalam properti bound.
- locale – lokasi yang digunakan
- maxIntegerDigits – angka maksimum dari digit yang akan ditunjukkan atau digunakan ketika menangani bagian integer dari angka
- maxFractionDigits – angka maksimum dari digit yang akan ditunjukkan atau digunakan ketika menangani bagian desimal dari angka
- minFractionDigits – angka minimum dari digit yang akan ditunjukkan atau digunakan ketika menangani bagian desimal dari angka
- minIntegerDigits – angka minimum dari digit yang akan ditunjukkan atau digunakan ketika menangani bagian integer dari angka
- pattern – pola yang digunakan ketika mem-format atau menampilkan angka

  Untuk lebih mendetail tentang pola yang disediakan, periksalah JavaDoc untuk java.text.NumberFormat
- type – mengindikasikan apakah angka yang diubah diberlakukan sebagai mata uang, persen, atau angka. Nilai yang mungkin adalah currency, percent, dan number. Untuk lebih detailnya, periksa JavaDoc untuk java.text.NumberFormat.

Salah satu hal yang penting dicatat dalam menggunakan NumberConverter seperti pada DateTimeConverter, beberapa pola atau simbol yang dinyatakan dalam atribut dijalankan sebagai aturan bagi user. Jika masukan user tidak mengikuti pola atau tidak menyatakan simbol yang diminta dalam masukannya, kesalahan konversi akan terjadi selanjutnya pemrosesan data akan berhenti.

Salah satu dari masalah dengan converter yang disediakan sebagai standar dengan JSF, mereka hanya menjalankan konversi standar atau mereka menjalankan pola masukan pada user. Untuk contohnya, dalam NumberConverter, jika kita menetapkan nilai dari 'P' dalam atribut currencySymbol, jika user tidak memasukkan 'P' yang mengikuti angka akan menyebabkan kesalahan/error.

Untungnya, JSF menyediakan cara mudah bagi developer untuk membuat converter sendiri untuk digunakan dalam framework.

10.4.3 Custom Converter

Custom converter dapat dibuat dengan membuat class yang mengimplementasikan interface Converter. Interface ini didefinisikan dengan dua method :

- public Object getAsObject(FacesContext ctxt, UIComponent component, String input)
- public Object getAsString(FacesContext ctxt, UIComponent component, Object source)

Mempertimbangkan skenario berikut : kita mempunyai halaman form yang meminta user untuk memasukkan jumlah moneter. Kita akan menyimpan masukan sebagai object Double sehingga memudahkan kemudahan pemrosesan nantinya. Bagaimanapun, field masukan harus cukup fleksibel untuk menangani masukan angka langsung (contoh : 2000), atau masukan yang memasukkan simbol yang berkelompok (2,000). Karena halaman akan kembali pada kontrol dirinya sendiri, data harus ditampilkan lagi sebagai simbol yang berkelompok.

Salah satunya akan berpikir bahwa karena operasi ini meliputi angka dan simbol pengelompokkan, kita dapat menyederhanakan penggunaan NumberConverter yang disediakan bersama framework. Bagaimanapun, NumberConverter menjalankan apapun pola yang disediakan padanya, sehingga jika kita diperintah menangani mata uang dengan simbol pengelompokkan, ini akan menangani HANYA mata uang dengan simbol pengelompokkan. Masukkan yang benar-benar angka akan menghasilkan kesalahan konversi. Juga tidak memungkinkan mempunyai banyak converter bagi komponen masukan tunggal, sehingga kita tidak dapat menyederhanakan penggunaan dua instance NumberConverter yang berbeda. Dengan skenario ini perhatikan bahwa contoh untuk custom converter dibuat.

10.4.4 Method getAsObject

Method ini dipanggil oleh framework pada konverter yang saling berhubungan ketika diperlukan untuk mengubah masukan user dari string yang merepresentasikannya ke dalam tipe object yang lain. Konsep dalam mengimplementasikan method ini adalah sederhana : laksanakan pemprosesan operasi pada parameter String yang disediakan dan kembalikan Object yang akan menampilkannya pada sisi server.

Mempertimbangkan gambaran skenario di atas, tugas kita dalam method ini adalah secara konsekwen mengubah masukan String ke dalam object Double, tanpa memperhatikan apakan masukan berupa angka sebenarnya, atau salah satu simbol pengelompokkan.

Implementasi dari method ini ditunjukkan di bawah ini :

```
   public Object getAsObject(FacesContext facesContext, UIComponent uIComponent,
     String str) {
        Double doubleObject = null;
        try {
            doubleObject = Double.valueOf(str);
        } catch (NumberFormatException nfe) {
            //if exception occurs, one probable cause is existence of grouping symbol

            //retrieve an instance of a NumberFormat object
            //and make it recognize grouping symbols

            NumberFormat formatter = NumberFormat.getNumberInstance();
            formatter.setGroupingUsed(true);

            // use the NumberFormat object to retrieve the numerical value of the input
            try {
                Number number = formatter.parse(str);
                if (number.doubleValue() <= Long.MAX_VALUE) {
                    Long value = (Long) number;
                    doubleObject = new Double(value.doubleValue());
                } else {
                    doubleObject = (Double)number;
                }

            } catch (ParseException pe) {
                // if code reaches this point, none of the supported formats were followed.
                // create an error message and display it to the user through the exception
object

                FacesMessage message = new FacesMessage("Unsupported format",
                        "This field supports only numbers (5000), or numbers with commas
(5,000)");
                throw new ConverterException(message);
            }
        }

        return doubleObject;
    }
```

10.4.5 Method getAsString

Method ini yang membuat Converter dua arah : ini memerintah representasi String dari data internal yang sesuai. Konsep dibelakang implementasi dari object ini lebih mudah daripada method getAsObject di atas : menampilkan jumlah yang disimpan dalam object Double sebagai angka engan simbol pengelompokkan. Apa yang membuat implementasi ini lebih mudah adalah kita tahu dengan pasti format masukan yang tepat.

Berikut implementasinya :

```
    public String getAsString(FacesContext context, UIComponent component, Object source) {
        Double doubleValue = (Double)obj;
        NumberFormat formatter = NumberFormat.getNumberInstance();
        formatter.setGroupingUsed(true);

        return formatter.format(doubleValue);
    }
```

10.4.6 Menggunakan Custom Converter

Setelah kita membuat custom converter, salah satu langkah pertama yang telah kita ambil adalah mengkonfigurasi framework sehingga dapat dikenal eksistensinya. Ini dilakukan dengan menambahkan catatan konfigurasi dalam faces-config.xml.

File faces-config.xml mendefinisikan struktur yang tepat untuk elemennya. Catatan konfigurasinya datang setelah semua masukan-masukan untuk validator, tetapi sebelum masukan apapun untuk bean.

Di bawah ini catatan konfigurasi untuk converter yang telah kita buat.

```
<converter>
    <description>Custom converter for accepting monetary input</description>
    <converter-id>myConverter</custom-id>
    <converter-class>jedi.sample.MyCustomConverter</converter-class>
</converter>
```

Setelah kita membuat catatan konfigurasi untuk converter kita, menggunakannya untuk elemen masukan semudah menetapkan id converter dalam tag <f:converter>, seperti :

```
<h:inputText id="amount" value="#{numberBean.amount}">
    <f:converter converterId="myConverter"/>
</h:inputText>
```