Tillægskapitel til "Webprogrammering med JSP"

Jacob Nordfalk

http://javabog.dk/

30-03-05

14 Java Server Faces

Dette kapitel er frivillig læsning; det forudsættes ikke i resten af bogen.

Blablabla

xxx JSF er et HTML-lignende sprog, som man kan skrive koden, der udføres på serveren i, i stedet for Java. For ikke-Java-kyndige HTML-designere skulle JSTL være betydeligt simplere at bruge end Java, da syntaksen ligner den, de i forvejen kender fra HTML[1].

JSF fungerer i praksis som to tag-biblioteker (eng.: taglibs). Det vis sige at JSP-sider der bruger JSF-komponenter skal have linjerne

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

i starten af siden.

Et tag library (forkortet taglib) er, som navnet siger, et bibliotek af HTML-lignende koder. Ligesom JSP-koderne udføres taglib-koderne på serveren.

Læsning om JSF

- Officiel side på Sun: http://java.sun.com/j2ee/javaserverfaces/
- http://jsftutorials.net
- Om JSF og JDeveloper: http://www.oracle.com/technology/products/jdev/jsf.html

14.1 Kernefunktionalitet (<f: >)

Kernefunktioner (de mest basale funktioner) i JSF ligger i core-taglibbet under navnet 'f'. Her er understøttelse for hændelser, datakonvertering, lokalisering, validatorer og andre ting.

Alle JSF-koder skal f.eks. være indeholdt i et 'view', der sørger for at registrere og konfigurere JSF-komponenterne:

```
<f:view>
  ... JSF-koder (og eventuelt andre JSP-koder)
</f:view>
```

De andre <f:>-koder angår alle kernefunktionalitet i JSF:

xxx oversigstabel ind

14.2 HTML-funktionalitet (<h: >)

JSF er beregnet til at kunne generere mange former for output (det mest almindelige er selvfølgelig HTML). Til dette bruges et JSF 'render kit', der er et tagbibliotek med komponenter beregnet til den pågældende form for output, f.eks. JSFs HTML-tagbibliotek <h:>:

xxx oversigstabel ind

1 JSP med JSTL kan minde om sproget Coldfusion fra Macromedia/Allaire

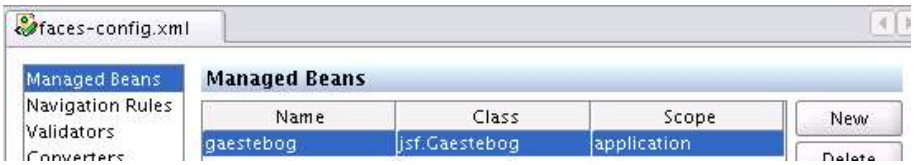
14.3 Visning af gæstebog fra JSF

Herunder ses hvordan man kunne lave gæstebogen vi så i databasekapitlet i afsnit xxx med JSF.

Bind et objekt (en javabønne), til applikationen, med følgende i faces-config.xml:

```
  <managed-bean>
    <managed-bean-name>gaestebog</managed-bean-name>
    <managed-bean-class>jsf.Gaestebog</managed-bean-class>
    <managed-bean-scope>application</managed-bean-scope>
  </managed-bean>
```

I JDeveloper (og sikkert også andre værktøjer) kan det gøres ved at gå hen på fanen 'Overview' på faces-config.xml:



Denne klasse har logikken til at hente et ResultSet med data, med metoden getGaester():

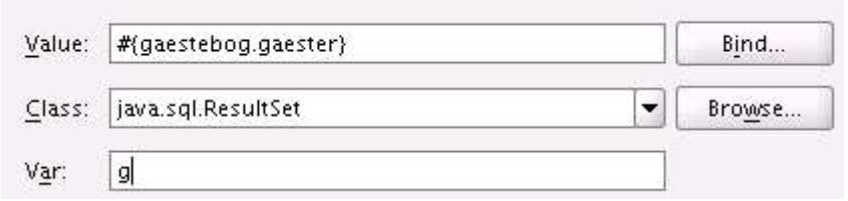
```
package jsf;
import java.sql.*;

public class Gaestebog
{
  private Connection con;

  public Gaestebog()
  {
    try {
      // Initialisering gæstebogen
      Class.forName("com.mysql.jdbc.Driver");
      con = DriverManager.getConnection("jdbc:mysql:///test");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public ResultSet getGaester()
  {
    System.out.println("inde i getGaester()");
    try {
      // Hent ResultSet med gæstebogen
      Statement stmt = con.createStatement();
     ResultSet rs = stmt.executeQuery("SELECT navn, besked, dato FROM gaestebog");
      return rs;
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }
  }
}
```

Nu kan vi lave en JSF-side med en <h:dataTable />, der viser gæsterne. Værktøjet spørger os om hvilken værdi der skal vises (det er #{gaestebog.gaester}, dvs metoden getGaester() på Gaestebog-objektet, der jo giver ResultSet-objektet) og om navnet på en variabel, der skal gennemløbe alle rækkerne.



Dernæst skal vi beskrive hvad der skal vises i hver kolonne. Da vi har kaldt vores gennemløbsvariabel for 'g' er det den vi skal bruge. Da kolonnerne i databasen hedder hhv 'navn', 'dato' og 'besked' skriver vi hhv #{g.navn}, #{g.dato} og #{g.besked} (egentlig bliver det oversat til getString("navn"), getString("dato") og getString("besked") på ResultSet'et):

Header Value	Component	Component Value
navn	Output Text	#{g.navn}
datoen	Output Text	#{g.dato}
beskedden	Output Link	#{g.besked}

Herefter ser siden nogenlunde sådan her ud:

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<f:view>
  <html>
    <head><title>Gæstebog JSF</title></head>
    <body>
      <h:form>
        <h1>Gæstebog-jsf</h1>
        <p>Velkommen til min lille gæstebog.</p>
        <p>
          <h:dataTable value="#{gaestebog.gaester}" var="g">
            <h:column>
              <f:facet name="header">
                <h:outputText value="navn"/>
              </f:facet>
              <h:outputText value="#{g.navn}"/>
            </h:column>
            <h:column>
              <f:facet name="header">
                <h:outputText value="datoen"/>
              </f:facet>
              <h:outputText value="#{g.dato}"/>
            </h:column>
            <h:column>
              <f:facet name="header">
                <h:outputText value="beskedden"/>
              </f:facet>
              <h:outputLink>
                <h:outputText value="#{g.besked}"/>
              </h:outputLink>
            </h:column>
            <h:column/>
          </h:dataTable>
        </p>

        <p>
          Du kan skrive dig ind
          <h:commandLink action="indskriv">
            <h:outputText value="her"/>
          </h:commandLink>
        </p>
```

```
      </h:form>

<%
  // Her ses hvordan man får fat i en JSF-bønne fra almindelig JSP-kode
  int antalGæster = 0;
  jsf.Gaestebog gæstebog = (jsf.Gaestebog) application.getAttribute("gaestebog");
  java.sql.ResultSet rs = gæstebog.getGaester();
  while (rs.next()) antalGæster++; // tæl antal rækker
  out.print("<p>Der er i alt "+antalGæster+" besøg.</p>");
%>
    </body>
  </html>
</f:view>
```

Gæstebog-jsf

Velkommen til min lille gæstebog.

navn	dato	besked
bubber	2005-03-08 00:00:00.0	hej verden!
javob	2005-03-29 00:00:00.0	hej igenigeb
Jacuib	2005-03-29 00:00:00.0	hejsa alle sammen
jacob2	2005-03-29 00:00:00.0	haaaaj
zsdsdsd	2005-03-29 00:00:00.0	sd.kshd

Du kan skrive dig ind her

Der er i alt 5 besøg.

Nederst ses hvordan man kan få fat i JSFs objekter fra almindelig JSP-kode:

```
  jsf.Gaestebog gæstebog = (jsf.Gaestebog) application.getAttribute("gaestebog");
```

Sammenlign med faces-config.xml, der erklærede at bønnenavnet var 'gaestebog' med virkefelt application af type ' jsf.Gaestebog'.

14.4 Opdatering af gæstebog fra JSF

Lad os nu lave en JSF-side, der opdaterer gæstebogen:

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<f:view>
  <html><head><title>Gæstebog - indskriv</title></head>
    <body>
      <h:form>
        <h:commandLink action="tilbage">
          <h:outputText value="tilbage til gæstebog"/>
        </h:commandLink>

        <h1>Skriv dig ind i min gæstebog</h1>
        <p>
            Navn: <h:inputText value="#{gaest.navnet}" />
        </p>
        <p>
          Besked:<br></br>
         <h:inputTextarea value="#{gaest.besked}" cols="70" rows="5" />
        </p>
        <p>
          <h:commandButton value="Ok" action="#{gaestebog.indskriv}"/>
        </p>
      </h:form>
    </body>
  </html>
</f:view>
```

xxx skærmbillede ind

Her har vi sat de to JSF-komponenter <h:inputText/> og <h:inputTextarea/> til at binde til egenskaberne hhv 'navnet' og 'besked' på javabønnen 'gaest', der styres af JSF.

14.4.1 Avanceret: Validatorer

Man kan tilføje nogle JSF-validatorer til input-komponenter.

Herunder kræver vi at brugeren indtaster et navn på mindst 4 og højst 20 tegn:

```
Navn: <h:inputText value="#{gaest.navnet}">
   <f:validateLength minimum="4" maximum="20"/>
</h:inputText>
```

Der findes selvfølgelig mere sofistikerede validatorer end <f:validateLength/>, der kun tjekker længden af input. Referenceimplementationen har kun <f:validateLongRange/> og <f:validateDoubleRange/> til at tjekke intervaller på hhv heltal og kommatal, men andre JSF-komponentbiblioteker (f.eks. Oracle ADF og Apache MyFaces, se afsnit 14.8, Kilder til JSF-komponenter), har en række nyttige validatorer til f.eks. at tjekke om en streng er gyldig som epost-adresse eller overholder et bestemt regulært udtryk (eng.: regular expression).

14.4.2 Avanceret: Fejlmeddelelser

Ovenstående tjekker længden men giver ingen fejlmeddelelse tilbage til brugeren i tilfælde af fejl. Til det skal man give ens <h:inputText/> et id og indsætte en <h:message/>-kode, der så vil vise fejlmeddelelsen (skal have en for-attribut med navnet på inputtekstens id).

```
Navn: <h:inputText value="#{gaest.navnet}" id="navnfelt">
   <f:validateLength minimum="4" maximum="20"/>
</h:inputText>
<h:message for="navnfelt" errorStyle="color:rgb(255,0,0);"/>
```

Her har vi også valgt at vise fejlteksten i rødt (errorStyle="color:rgb(255,0,0);").

Fejlmeddelserne er desværre ikke særligt sigende og dertil på engelsk. Det er muligt at skrive sine egne fejlmeddelelser, men på nuværende tidspunkt (april 2005) kun ved at lægge dem ud i separate tekstfiler (kaldet resursebundter eller .properties-filer)

14.4.3 Avanceret: Resursebundter

Se afsnit 13.4, Tekstindhold i resursefiler.

Fra JSF aktiverer man et resursebundt med <f:loadBundle/> og angiver resursebundtet og den variabel det skal være tilgængeligt i. Følgende indlæser bundtet 'sprogtest.Tekster' og gemmer i JSP-variablen 'tekster':

```
<f:loadBundle basename="sprogtest.Tekster" var="tekster"/>
```

Herefter kan 'tekster' bruges som en hvilken som helst anden variabel. F.eks. kan teksten under nøglen 'Kl_' udskrives med:

```
<h:outputText value="#{tekster.Kl_}"/>
```

Xxx uddyb - og bedre eksempel... måske den internationale kalender?

14.4.4 At gemme data i en javabønne

Vi skal nu skrive klassen, der har egenskaberne hhv 'navnet' og 'besked'. Den ligger i pakken 'jsf' og kaldes Gaest:

```
package jsf;

public class Gaest
{
  String navnet;
  String besked;

  public String getNavnet()
  {
    return navnet;
  }

  public void setNavnet(String navnet)
  {
    this.navnet = navnet;
  }

  public String getBesked()
  {
    return besked;
  }

  public void setBesked(String besked)
  {
    this.besked = besked;
  }
}
```

Bemærk at de fleste udviklingsværktøjer kan generere get- og set-metoderne for os, sådan at vi slipper for at spilde tid med at indtaste dem selv. Det er dog stadig et kedeligt arbejde at generere og vedligeholde dem (senere, i afsnit 14.6.1At gemme midlertidige data i en HashMap) vil vi se et bud på hvordan man kan undgå at skrive alt for meget af denne slags 'hjernedød' kode).

Til sidst kan vi nu binde klassen 'jsf.Gaest' til navnet 'gaest' i faces-config.xml:

```
  <managed-bean>
    <managed-bean-name>gaest</managed-bean-name>
    <managed-bean-class>jsf.Gaest</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
```

Hermed er javabønnen bundet og tilgængelig i JSP-siderne.

14.4.5 Avanceret: Startværdier for egenskaber

Somme tider kan det være rart at kunne angive nogle startværdier som egenskaberne på javabønnerne sættes til ved oprettelsen (eng.: managed properties). Det gøres ved at indsætte en <managed-property>-kode i beskrivelsen af bønnen:

```
  <managed-bean>
    <managed-bean-name>gaest</managed-bean-name>
    <managed-bean-class>jsf.Gaest</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
    <managed-property>
      <property-name>navnet</property-name>
      <property-class>java.lang.String</property-class>
      <value>anonym</value>
    </managed-property>
  </managed-bean>
```

Herefter vil gæsten have have navnet 'anonym' indtil han indtaster noget andet.

14.5 Aflæsning af bønnens egenskaber

Vi mangler at aflæse vores gaest-bønne og sætte værdierne ind i databasen. Koden til knappen, der skal aktivere indskrivning ser således ud:

```
          <h:commandButton value="Ok" action="#{gaestebog.indskriv}"/>
```

Det betyder at vi skal have metoden public String indskriv() i Gaestebog-objektet. Denne metode afgør - sammen med faces-config.xml - hvilken side der gås hen til når brugere trykker på knappen.

Her ses metoden indskriv(), der finder gaest-bønnen frem og aflæser dens egenskaber:

```
...
public class Gaestebog
{
  private Connection con;
  ...

  public String indskriv()
  {
    System.out.println("indskriv()");
    try {
      FacesContext fc = FacesContext.getCurrentInstance();

      Gaest g = (Gaest) fc.getExternalContext().getRequestMap().get("gaest");

      PreparedStatement pstmt = con.prepareStatement(
        "INSERT INTO gaestebog (navn, besked, dato, ip) VALUES(?,?,?,?)");

      pstmt.setString(1, g.getNavnet());
      pstmt.setString(2, g.getBesked());
      pstmt.setDate(3, new java.sql.Date(System.currentTimeMillis()));

      HttpServletRequest request;
      request = (HttpServletRequest) (fc.getExternalContext().getRequest());

      pstmt.setString(4, request.getRemoteAddr());
      pstmt.executeUpdate();
      pstmt.close();
      con.close();

      return "tilbage";
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }
  }
}
```

Linjen

```
      FacesContext fc = FacesContext.getCurrentInstance();
```

er central. Den giver os adgang til hele JSF-systemet, der ligger som et lag oven på JSP.

I dette eksempel er vi dog kun interesserede i det underliggende JSP-system (JSF-systemets 'ExternalContext'), navnlig request-objektet (gaest-bønnen er har virkefelt 'request') og kalder derfor

```
fc.getExternalContext().getRequest()
```

Fordi JSF i princippet kan fungere oven på andre systemer end JSF er returværdien af denne metode blot 'Object' og vi må selv konvertere den til et HttpServletRequest-objekt.

Adgang til session-objektet fra JSF

Man kan få adgang til session-objektet (og de andre implicitte objekter - og deres objekter) med f.eks.

```
FacesContext fc = FacesContext.getCurrentInstance();
HttpSession session = (HttpSession) fc.getExternalContext().getSession(false);
```

14.6 JSF - anbefalet praksis

Jeg anbefaler at man binder forretningsdata, data af lidt mere permanent karakter og data hvor noget lidt mere kompleks programlogik er involveret til javabønners egenskaber som beskrevet ovenfor.

14.6.1 At gemme midlertidige data i en HashMap

Til midlertidige formulardata, der bare nemt skal opbevares et sted indtil de skal behandles eller gemmes et andet sted (f.eks. i en database som Gaest-klassen ovenfor) synes jeg dog det bliver for tungt at oprette og vedligeholde alle disse javabønne. I disse tilfælde kan man bare bruge en HashMap (en nøgleindekseret tabel, se afsnit 14.10) til opbevaringen.

Herunder ses ovenstående eksempel *uden* brug af Gaest-klassen. I stedet bruges en HashMap (sammenlign med tidligere erklæring af bønnen 'gaest' i faces-config.xml):

```
  <managed-bean>
    <managed-bean-name>gaest</managed-bean-name>
    <managed-bean-class>java.util.HashMap</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
```

Vi behøver ikke ændre på selve JSF-siden, da udtrykket

```
<h:inputText id="navn" value="#{gaest.navn}" />
```

automatisk bliver opfattet anderledes af JSF hvis gaest er en HashMap: I stedet for af forvente metoderne String getNavnet() og void setNavnet(String navnet) kaldes nu metoderne get("navnet") og set("navnet", navnet), dvs brugerens indtastning bliver gemt under nøglen "navnet" i hashtabellen.

Her ses den endelige kode for Gaestebog (ændringer i forhold til tidligere versioner er i fed):

```
package jsf;
import java.sql.*;
import java.util.HashMap;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletRequest;

public class Gaestebog
{
  private Connection con;

  public Gaestebog()
  {
    try {
      // Initialisering gæstebogen
      Class.forName("com.mysql.jdbc.Driver");
      con = DriverManager.getConnection("jdbc:mysql:///test");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
```

```
  public ResultSet getGaester()
  {
    System.out.println("getGaester()");

    try {
      // Udskriv gæstebogen
      Statement stmt = con.createStatement();
     ResultSet rs = stmt.executeQuery("SELECT navn, besked, dato FROM gaestebog");
      return rs;
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }
  }

  public String indskriv()
  {
    System.out.println("indskriv()");
    try {
      FacesContext fc = FacesContext.getCurrentInstance();

      System.out.println("fc="+fc);

      HashMap g = (HashMap) fc.getExternalContext().getRequestMap().get("gaest");
      System.out.println("formular="+g);

      String navn = (String) g.get("navn");
      String besked = (String) g.get("besked");

      PreparedStatement pstmt = con.prepareStatement(
        "INSERT INTO gaestebog (navn, besked, dato, ip) VALUES(?,?,?,?)");

      pstmt.setString(1, navn);
      pstmt.setString(2, besked);
      pstmt.setDate(3, new java.sql.Date(System.currentTimeMillis()));

      HttpServletRequest request;
      request = (HttpServletRequest) (fc.getExternalContext().getRequest());

      pstmt.setString(4, request.getRemoteAddr());
      pstmt.executeUpdate();
      pstmt.close();
      con.close();

      return "tilbage";
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }
  }
}
```

14.7 Resumé

- JSF kan styre javabønner ('managed beans')
  - Styres i faces-config.xml
  - Ingen <jsp:useBean /> fra JSP-siderne mere
- Fire mulige virkefelter (eng.: scope): Request, session, application, ...
- JSF-komponenters værdier kan bindes til egenskaber

- Dette opfattes som egenskaber:
- getEgenskabsnavn()/setEgenskabsnavn(værdi)-metoder på objekt
- get(String egenskabsnavn)/put(String egenskabsnavn, værdi)
  - => mere fleksible bindinger
  - man kan også binde til værdier i datastrukturer (f.eks. HashMap) og databaser (f.eks. ResultSet)

14.7.1 JSF og JDeveloper

JDevelopers server (OC4J) ser ud til at have problemer med 'converters', dvs. JSF-koder, der konverterer valutaer, tal eller datoer. Det gør det svært at køre ret mange af de eksempler man finder på nettet.

14.7.2 Måder at aktivere JSF-komponenter

Der er flere måder at styre JSF-komponenterne og dens værdier.

1. I JSP-siden binde værdien (den attribut der hedder 'value') til egenskab i javabønne:
   F.eks. <h:inputText value="#{gaest.navnet}" /> hvor javabønnen har egenskaben 'navnet' af type String, dvs den har metoderne
   public String getNavnet() og
   public void setNavnet(String navnet)
2. I JSP-siden binde selve komponent-objektet (den attribut der hedder 'binding') til egenskab i javabønne:
   F.eks. <h:inputText binding="#{backing_gaestebog.inputText1}" /> hvor javabønnen har egenskaben 'inputText1' af type javax.faces.component.html.HtmlInputText, dvs den har metoderne
   public HtmlInputText getInputText1() og
   public void setInputText1(HtmlInputText inputText1)
3. Hver JSP-side er i JSF repræsenteret ved et træ af objekter. Dette objekttræ kan man ændre i og slette, ændre og tilføje nye komponenter. Det er således muligt at slå komponenten op og direkte aflæse/manipulere dens værdi fra Java-programkoden.

Det anbefales at holde sig til mulighed 1. JDeveloper genererer automatisk (det kan heldigvis slås fra) 'backing beans' til mulighed 2.

14.8 Kilder til JSF-komponenter

Som bekendt er JSF opdelt i en generel del ('core'), der ikke er specielt rettet mod HTML og en HTML-specifik del, der indeholder de visuelle komponenter.

- Sun JSF RI (referenceimplementationen)
- Oracle ADF Faces
- Apache MyFaces

14.8.1 Oracle ADF Faces

Oracle ADF Faces (Application Development Framework) er er Oracles egne webkomponenter skrevet om til JSF. Det er en en imponerende samling af omkring 100 komponenter, selvfølgelig indbefattet en række der kan vise data fra en database.

Læs mere om ADF på http://www.oracle.com/technology/products/jdev/index.html under 'Oracle ADF --> Online tour'

14.8.2 Apache MyFaces

Apache MyFaces er en samling JSF-komponenter med Åben Kildekode (Open Source).

Se en række eksempler på MyFaces-komponenter her (klik på Examples og Components). Prøv f.eks. komponenterne File Upload, Calendar, HTML Editor og Tree.

Der er ca. 25 komponenter og validatorer. Kør selv den nyeste udgave af eksemplerne (de findes også i en simplere udgave, der er lettere at forstå) for at se alle komponenterne.

WAR-filen kan køres direkte i Tomcat (smid den i webapps/) eller fra JDeveloper (opret projekt fra WAR-fil). Du starter eksemplerne ved at køre 'index.jsp'.

Yderligere læsning

MyFaces' hjemmeside: http://myfaces.org/ (nyeste udgave er på http://myfaces.apache.org/).

MyFaces med JDeveloper

Som det fremgår af

http://www.oracle.com/technology/products/jdev/101/howtos/myfaces/index.html
kan man desværre ikke installere MyFaces i JDevelopers komponentpalette i betaudgaven:

”In the JDeveloper 10.1.3 Developer Preview developers are restricted in that they cannot switch out the provided JSF implementation with another implementation such as MyFaces. In the production release of JDeveloper 10.1.3 developers will be able to switch out the by default provided RI and use a JSF implementations of their choice.”

Man kan dog sagtens køre og redigere MyFaces-komponenter.

14.8.3 Andre kilder

- Ourfaces - https://ourfaces.dev.java.net (åben kildekode): Træ, Tabel, Kalender. Opfordring til flere bidrag
- Chart FX for Java - http://eu.softwarefx.com/SFXJavaProducts/CFXforJava/ (lukket kildekode): Alle former for 2D- grafer
- ILOG JViews Charts - http://ilog.com/products/jviews/charts/ (lukket kildekode): Alle former for 2D- og 3D- grafer
- Otrix - http://otrix.com/products/ (lukket kildekode): Menu, Træ, Tabel
- WebCharts 3D - http://www.gpoint.com (lukket kildekode): 2D- og 3D-datavisualiseringskomponenter
- WebGalileo Faces Components - http://jscape.com/webgalileofaces/ (lukket kildekode): Tabbed Panel, Toolbar, Menu, Tree, Table, Pop-Up Dialog, HTML Editor, Calendar, Color Dialog, Calculator, Tree Table

14.9 EL - Expression Language

Værdier fra variabler og regneudtryk skal i JSF puttes ind i et #{}-udtryk (i JSTL er det ${}), som er en måde at gøre brug af EL - Expression Language.

EL er et elegant sprog, der gør det muligt, at skrive ret lange Java-udtryk på kort form.

Således kan knudrede Java-udtryk, som f.eks.:

```
Velkommen <%= ((Bruger)session.getAttribute("bruger")).getNavn() %> !
```

med EL skrives som blot (xxx tjek):

```
Velkommen ${session.bruger.navn} !
```

Det skyldes, at EL har en meget fleksibel punktum-notation, der automatisk undersøger egenskaber på javabønner og gennemsøger attributter på implicitte objekter og andre nøgleindekserede objekter (hashtabeller).

14.10 Om associative tabeller (HashMap)

Xxx råtekst fra andre bøger - skal kortes ned

En hashtabel er en tabel, der afbilder nogle nøgler til værdier. Hashtabeller er nyttige som associative afbildninger - når man vil indeksere nogle objekter ud fra f.eks. navne.

Man opretter en hashtabel med:

```
  HashMap tabel = new HashMap();
```

Derefter kan man lægge en indgang i tabellen med **put**(nøgle, værdi). F.eks.:

```
  tabel.put("abc", "def");
```

husker strengen "def" under nøglen "abc". Vil vi finde strengen frem igen, skal vi slå op under "abc":

```
  String værdi = (String) tabel.get("abc");
```

Da hashtabeller oftest anvendes til at lave afbildninger med, bruger man i dagligdags sprogbrug mere ordet 'hashtabel' end ordet 'afbildning'.

Før vi går videre, så bemærk lige, at lister *går fra heltal til objekter*, dvs. man finder listens elementer frem ud fra et helt tal (indekset). Blandt andet har en liste metoderne:

void **add**(int indeks , element)
  Indsætter *element* i listen lige før plads nummer *indeks*. Første element er på plads nummer 0.

Elementtype **get**(int indeks)
  returnerer en reference til objektet på plads nummer *indeks*.

Man kan sige, at elementerne i en liste indekseres (fremfindes) ud fra et *tal*.

Hashtabeller *går fra objekter til objekter* på den måde, at til hvert element knyttes et nøgleobjekt. Elementerne kan derefter findes frem ud fra nøglerne.

Man kan altså sige, at elementerne i en hashtabel indekseres (fremfindes) ud fra et *objekt*.

java.util.HashMap - nøgleindekseret tabel af objekter

Konstruktører

HashMap<Nøgletype, Elementtype> ()
opretter en tom tabel hvor nøglerne er af klassen *Nøgletype* og værdierne af klassen *Elementtype*. <Nøgletype, Elementtype> kan udelades.

Metoder

void **put** (Nøgletype nøgle, Elementtype værdi)
føjer objektet *værdi* til hashtabellen under objektet *nøgle*.

Elementtype **get** (Nøgletype nøgle)
slår op under *nøgle* og returnerer den værdi, der findes der (eller null hvis nøglen ikke kendes).

Elementtype **remove**(Nøgletype nøgle)
fjerner indgangen under *nøgle* og returnerer værdien (eller null hvis nøglen ikke kendes).

boolean **isEmpty**()
returnerer sand, hvis tabellen er tom (indeholder 0 indgange).

int **size**()
returnerer antallet af indgange.

boolean **containsKey**(Nøgletype nøgle)
returnerer sand, hvis *nøgle* findes blandt nøglerne i tabellen.

boolean **containsValue**(Elementtype værdi)
returnerer sand, hvis *værdi* findes blandt værdierne i tabellen.

Set<Nøgletype> **keySet**()
giver alle nøglerne. Kan bruges til at gennemløbe alle indgangene (se nedenfor).

String **toString** ()
giver tabellens indhold som en streng. Dette sker ved at konvertere hver af indgangenes nøgler og værdier til strenge.

Herunder opretter vi en tabel, der holder styr på fødselsdatoer for et antal personer med deres fornavne som nøgler, med put()-metoden: put("Jacob", dato). Derefter kan indgangene hentes tilbage igen med get()-metoden: get("Jacob") giver Jacobs fødselsdato.

Sidst gennemløbes alle indgangene. Vi bruger en for-løkke til at løbe gennem tabellens nøgler (med hashtabel.keySet()), hvorefter vi slår de tilsvarende værdier op[1].

```
import java.util.*;
public class BenytHashMap
{
  public static void main(String[] arg)
  {
    // En tabel med strenge som nøgler og Date-objekter som værdier
    HashMap<String,Date> hashtabel = new HashMap<String,Date>();
    Date dato = new Date(71,0,1); // 1. januar 1971
    hashtabel.put("Jacob",dato);
    hashtabel.put("Troels",new Date(72,7,11)); // 11. august 1972
    hashtabel.put("Eva",new Date(73,2,5));
    hashtabel.put("Ulla",new Date(69,1,9));
    System.out.println( "tabel indeholder: "+hashtabel );

    // Lav nogle opslag i tabellen under forskellige navne
    dato = hashtabel.get("Troels");
    System.out.println( "Opslag under 'Troels' giver: "+dato);
    System.out.println( ".. og under Jacob: "+hashtabel.get("Jacob"));
    System.out.println( ".. Kurtbørge: "+hashtabel.get("Kurtbørge"));
```

1 Bruger du JDK 1.4 eller tidligere skal du fjerne <String,Date> fra new HashMap og bruge en iterator, som skitseret i kommentaren nederst. Iterator-objekter har to metoder: hasNext(), der fortæller, om der er flere elementer og next(), der går videre til næste element og returnerer det.

```
    System.out.println( ".. Eva: "+hashtabel.get("Eva"));

    // Gennemløb af alle elementer
    for (String nøgle : hashtabel.keySet()) {
      dato = hashtabel.get(nøgle);
      System.out.println(nøgle + "'s fødselsår: "+dato.getYear());
    }
    //JDK1.4: for (Iterator i = hashtabel.keySet().iterator(); i.hasNext();) {
      //JDK1.4: String nøgle = (String) i.next();
  }
}
```

```
tabel indeholder: {Jacob=Fri Jan 01 00:00:00 CET 1971, Troels=Fri Aug 11 00:00:00
CET 1972, Eva=Mon Mar 05 00:00:00 CET 1973, Ulla=Sun Feb 09 00:00:00 CET 1969}
Opslag under 'Troels' giver: Fri Aug 11 00:00:00 CET 1972
.. og under Jacob: Fri Jan 01 00:00:00 CET 1971
.. Kurtbørge: null
.. Eva: Mon Mar 05 00:00:00 CET 1973
Jacob's fødselsår: 71
Troels's fødselsår: 72
Eva's fødselsår: 73
Ulla's fødselsår: 69
```

En hashtabel husker ikke rækkefølgen af indgangene. Derfor er rækkefølgen, som elementerne bliver udskrevet i, ikke den samme som den rækkefølge, de blev sat ind i.

Her er en lille esperanto-dansk-ordbog. Nøglerne er esperanto og værdierne er danske ord:

```
import java.util.*;
public class BenytHashMapOrdbog
{
  public static void main(String[] args)
  {
    HashMap<String,String> ord = new HashMap<String,String>();
    ord.put("granda", "stor");
    ord.put("longa", "lang");
    ord.put("bela", "smukt");
    ord.put("estas", "er");

    String esperantotekst = "longa, granda hundo estas.... bela!";

    for (String eoOrd : esperantotekst.split("\\b")) { // split efter ordgrænser
      String da = ord.get( eoOrd ); // slå esperantoord op og få det danske ord
      if (da == null) da=eoOrd;  // hvis intet fundet lader vi det stå uoversat
      System.out.print( da );
    }
  }
}
```

```
lang, stor hundo er.... smukt!
```

Har vi en tekst på esperanto, kan vi nu oversætte teksten ord for ord til dansk. Hvert ord slås op i hashtabellen og hvis det findes, erstattes det med det danske ord. Tegn og ord, som ikke kan findes i tabellen (såsom "hundo", der betyder "hund"), efterlades uforandret.