Web- og serverprogrammering

Java Server Pages - dag 5

Avanceret JSP
Understøttelse for flere klienter
Tråde og synkronisering
Servletter og JSP-siders interne virkemåde
Læsning: WJSP 2.8 - 2.8.6, 3.6-3.7, 4.3 - 4.6, 7.1, 7.3

Spørgsmål fra "salen"

- Kan du fortælle lidt mere om brug at de 2 include direktiver - hvad er forskellen, kan de 'nestes', hvordan anvendes include i forbindelse med servlets?

`<%@ include ... %>`	Inklusions-direktivet. Inkluderer en fil (som om dens indhold blev klistret ind her - se afsnit 4.2.1 Inkludering af kodefragmenter). Eksempel `<%@ include file="hej.jsp" %>`
`<jsp:include ... />`	Kald en anden fil på køretidspunktet (som om der blev lavet en forespørgsel på den) og inkluderer dens uddata her, f.eks.: `<jsp:include page="hej.jsp"/>`

<%@ include file="logintjek.jsp" %>

- Statisk inklusion
  - Kildekode i logintjek.jsp kopieres ind i JSP-siden før oversættelse

<jsp:include page="logintjek.jsp" />

- Dynamisk inklusion
  - Hvad der skal inkluderes afgøres på anmodningstidspunktet
    <% String side="logintjek.jsp"; %>
    <jsp:include page="<%= side %>" />

Dynamisk inklusion kan også ske fra en servlet:
request.getRequestDispatcher("logintjek.jsp").include(request,response);

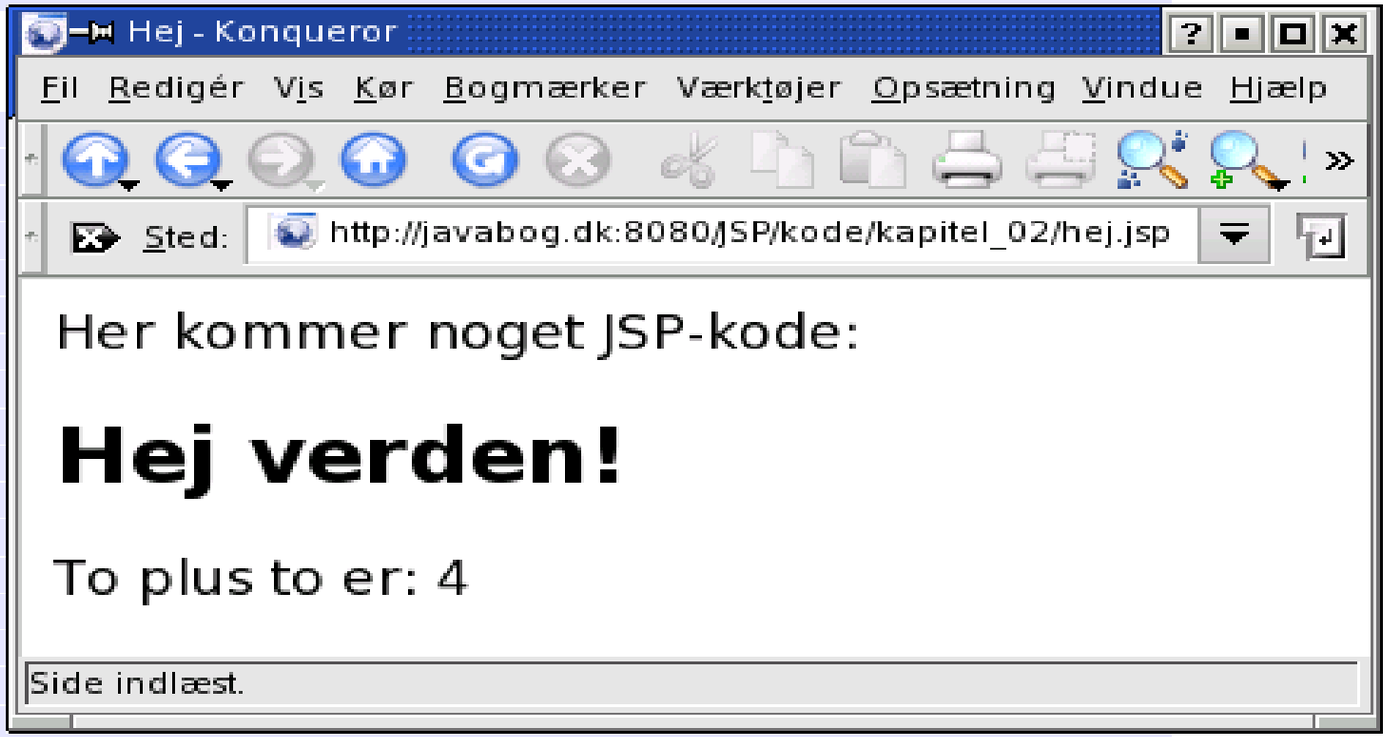
HTTP-protokollen

Klienten forbinder sig til serveren og sender:

```
GET /JSP/kode/kapitel_02/hej.jsp HTTP/1.1
Host: javabog.dk:8080
User-Agent: Mozilla/5.0 (X11; U; Linux i686; da-DK; rv:1.6) Gecko/20040115
Accept: text/xml,text/html,text/plain,image/png,image/jpeg,image/gif,*/*
Accept-Language: da,eo;q=0.8,no;q=0.5,sv;q=0.3
Referer: http://javabog.dk:8080/JSP/kode/kapitel_02/
```

Serveren svarer:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 130
Date: Thu, 05 Aug 2004 23:53:25 GMT
Server: Apache-Coyote/1.1

<html>
<head><title>Hej</title></head>
<body>
Her kommer noget JSP-kode:

<h1>Hej verden!</h1>
To plus to er:
4

</body>
</html>
```



Generere grafik fra JSP-side

- En side med MIME-typen ”image/jpeg”, behandles som et grafisk billede
  - http://localhost/billede.jsp

  - http://localhost/billede.jsp?tekst=JSP+er+nemt

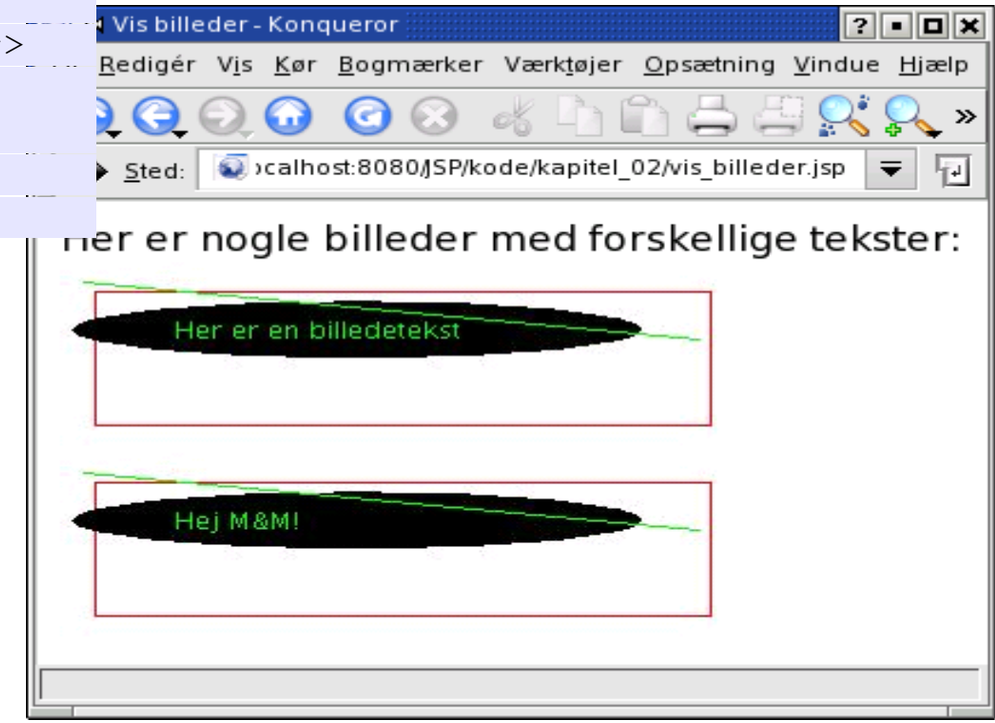
- For at vise en webside med et servergenereret billede
  - Lav JSP-side med HTML, der indeholder en henvisning til billedet, f.eks. <img src="billede.jsp">, i HTML-koden
  - Lav billede.jsp, der har MIME-typen ”image/jpeg” med billeddata

billede.jsp

```
<%@ page import="java.awt.*,java.awt.image.*,com.sun.image.codec.jpeg.*" %>
<%@ page contentType="image/jpeg" %>
<%
 BufferedImage billede = new BufferedImage(300, 100,
BufferedImage.TYPE_INT_RGB);
 Graphics2D g = billede.createGraphics();

 g.setColor(Color.white);   // udfyld baggrund
 g.fillRect(0,0,300, 100);

 g.setColor(Color.red);             // tegn ramme
 g.drawRect(15, 15, 270, 70);

 g.setColor(Color.black);   // tegn nogle andre ting
 g.fillOval(5,20,250,30);
 g.setColor(Color.green);
 g.drawLine(10,10,280,40);

 // skriv tekst, ud fra parameter eller sessionsattribut "billedtekst"
 String tekst = request.getParameter("billedtekst");
 if (tekst==null) tekst = (String) session.getAttribute("billedtekst");
 if (tekst==null) tekst = "Hej grafiske verden!"; // standardtekst
 g.drawString(tekst,50,40);

 g.dispose();               // færdig med at tegne, frigiv grafik-objektet

 // og konvertér billede til JPG-format og send billedet
 ServletOutputStream sos = response.getOutputStream();
 JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(sos);
 encoder.encode(billede);
%>
```

Generere grafik fra JSP-side

```
<html>
<head><title>Vis billeder</title></head>
<body>
Her er nogle billeder med forskellige tekster:<br>
<img src="billede.jsp?billedtekst=Her+er+en+billedetekst"><br>
<img src="billede.jsp?billedtekst=<%=URLEncoder.encode("Hej
M&M!")%>"><br>
</body>
</html>
```



Indlejre billeder fra harddisken


```
// hent billedet fra harddisken
String sti = application.getRealPath("/kode/kapitel_01/jacob.jpg");
InputStream is = new FileInputStream(sti);
Image jacob = JPEGCodec.createJPEGDecoder(is).decodeAsBufferedImage();

// tegn billedet nedskaleret
g.drawImage(jacob,210,-30,70,120,null);
```

Servletter

- En servlet er en Java-klasse der bliver brugt i en webserver.
  - Servletten skal arve fra HttpServlet
  - Servletten skal have en doGet(req, resp)-metode
    (eller en doPost()-metode)
  - Response-objektet bruges til at skrive HTML-kode i.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpelServlet extends HttpServlet
{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response) throws IOException
  {
    response.setContentType("text/html");
    PrintWriter ud = response.getWriter();
     ud.println("<html>");
     ud.println("<head><title>Hej verden</title></head>");
     ud.println("<body>");
     ud.println("<h3>Hej verden!</h3>");
     ud.println("Simpelt eksempel på en servlet");
     ud.println("</body>");
     ud.println("</html>");
   }
}
```

Servletter

- Opsætning af servlet i web.xml

1. Navnet på servletten i <servlet-name>
2. Klassenavnet (incl. pakkenavn) i <servlet-class>
3. Hvilke(n) URL(er) på serveren der skal omdirigeres til servletten i <url-pattern> i en <servlet-mapping>

```
<web-app>
  ...
   <servlet>
     <servlet-name>En simpel servlet</servlet-name>
     <servlet-class>SimpelServlet</servlet-class>
   </servlet>
   <servlet-mapping>
     <servlet-name>En simpel servlet</servlet-name>
     <url-pattern>/servlet/SimpelServlet</url-pattern>
   </servlet-mapping>
  ...
</web-app>
```

Nu kan servletten ses på http://maskine.dk/servlet/SimpelServlet

Servletter

- <url-pattern> giver en række muligheder
  - Flere URL'er til det samme:

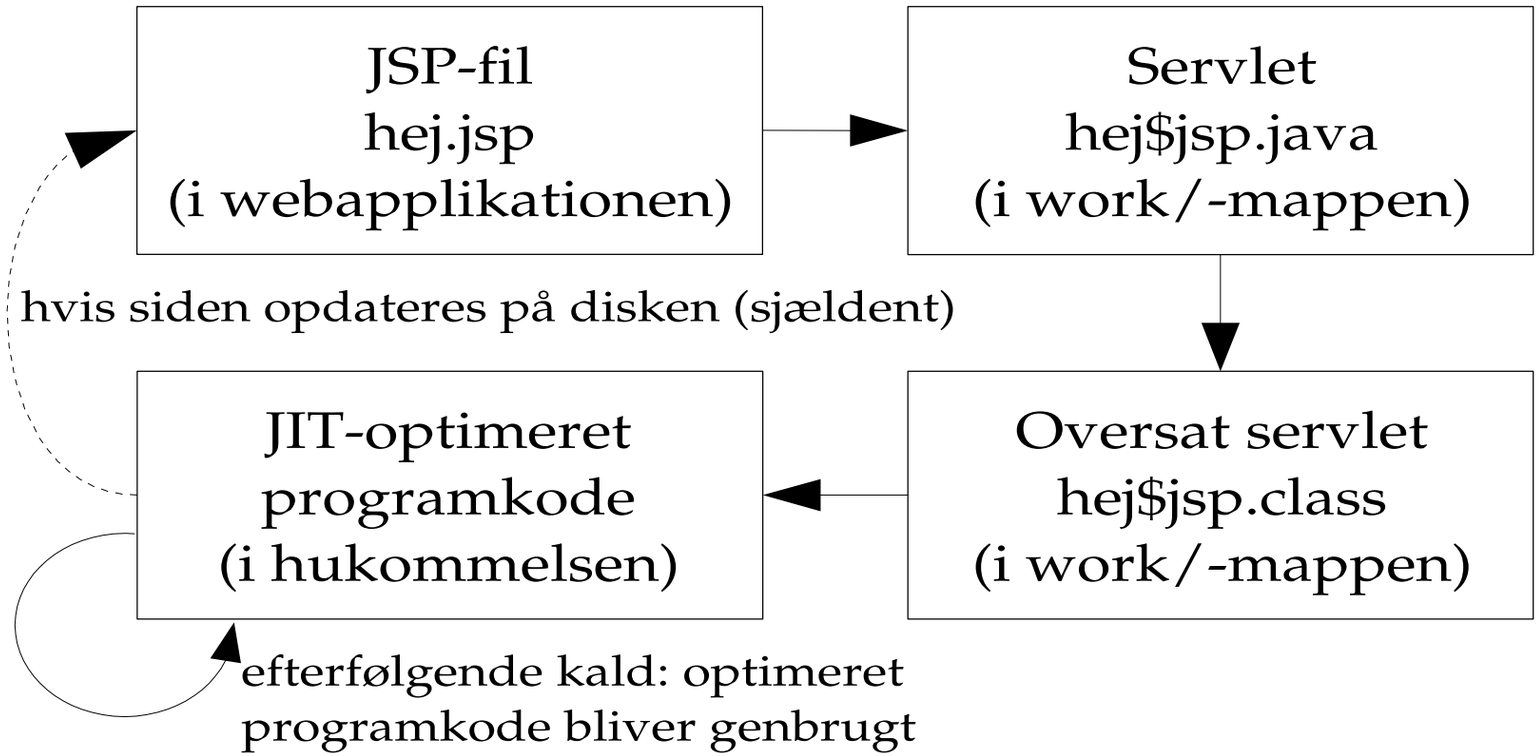
```
<web-app>
  ...
   <servlet>
     <servlet-name>En simpel servlet</servlet-name>
     <servlet-class>SimpelServlet</servlet-class>
   </servlet>
   <servlet-mapping>
     <servlet-name>En simpel servlet</servlet-name>
     <url-pattern>/simpel/*</url-pattern>
   </servlet-mapping>
  ...
</web-app>
```

  - automatisk binding af servletter
    - frarådes i drift (sikkerhedsproblemer)

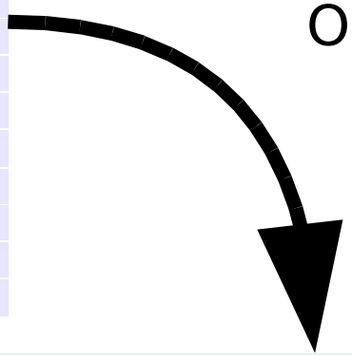
```
   <servlet>
     <servlet-name>invoker</servlet-name>
     <servlet-class>org.apache.catalina.servlets.InvokerServlet</servlet-class>
   </servlet>

   <servlet-mapping>
     <servlet-name>invoker</servlet-name>
     <url-pattern>/servlet/*</url-pattern>
   </servlet-mapping>
```

JSP-siders livscyklus

- JSP-sider oversættes internt til servletter
  - Langsomt første gang
  - Hurtig udførelse efterfølgende
  - Fejlmeddelelser ikke altid nemme at forstå



```
<html>
<head><title>Syvtabellen</title></head>
<body>
<p>Her er syv-tabellen:<br>

<%
  for (int i=1; i<=10; i++)
  {
%>
    Syv gange <%= i %> er: <%= 7*i %>.<br>
<%
  }
%>
</p>
</body>
</html>
```



Oversættes internt til

```
package org.apache.jsp.kode.kapitel_02;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class syvtabellen_jsp extends org.apache.jasper.runtime.HttpJspBase {
  public void _jspService( HttpServletRequest request,
                           HttpServletResponse response)
      throws java.io.IOException, ServletException {
    JspFactory _jspxFactory = null;
    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;

    try {
      _jspxFactory = JspFactory.getDefaultFactory();
      response.setContentType("text/html");
```

```
<html>
<head><title>Syvtabellen</title></head>
<body>
<p>Her er syv-tabellen:<br>

<%
  for (int i=1; i<=10; i++)
  {
%>
   Syv gange <%= i %> er: <%= 7*i %><br>
<%
  }
%>
</p>
</body>
</html>
```

```
    response.setContentType("text/html");
    pageContext = _jspxFactory.getPageContext(this, request, response,
                  null, true, 8192, true);
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
    _jspx_out = out;

    out.write("<html>\n");
    out.write("<head><title>Syvtabellen</title></head>\n");
    out.write("<body>\n");
    out.write("<p>Her er syv-tabellen:<br>\n");
    out.write("\n");

  for (int i=1; i<=10; i++)
  {
    out.write("\n");
    out.write("\t\tSyv gange ");
    out.print( i );
    out.write(" er: ");
    out.print( 7*i );
    out.write(".<br>\n");
  }

    out.write("\n");
    out.write("</p>\n");
    out.write("</body>\n");
    out.write("</html>");
  } catch (Throwable t) {
    if (!(t instanceof SkipPageException)){
      out = _jspx_out;
      if (out != null && out.getBufferSize() != 0)
        out.clearBuffer();
      if (pageContext != null) pageContext.handlePageException(t);
    }
  } finally {
    if (_jspxFactory != null) _jspxFactory.releasePageContext(pageContext);
  }
 }
}
```

Understøttelse for flere klienter

- Tråde og synkronisering
  - Lokale variabler OK
  - Objektvariabler (og static) variabler ikke OK

  - request - anmodningen fra klienten
  - response - svaret til klienten
  - out - skrive tekst til klienten
  - session - objekt der følger den enkelte bruger
  - application - fælles for hele webapplikationen
    - logning
    - konfigurations-parametre fra web.xml
    - kan også gemme attributter ligesom session-objektet
  - config - den enkelte websides konfiguration
  - page - selve JSP-siden
  - exception - undtagelse opstået under kørsel
  - pageContext - alle objekterne samlet i ét

Model 1 og 2 webarkitektur

- Model 1 - programlogik sammen med HTML
  - Simpel struktur
  - Nem at starte med
  - Velegnet til små projekter
  - Svært at adskille programlogik og HTML
  - Samme person er programmør og HTML-designer
  - Decentral
    - hver side behandler data fra sin egen formular
  - Potentiel redundans
    - samme programlogik flere steder
- Model 2 - programlogik adskilt fra HTML
  - Mere omfattende struktur
  - Sværere at starte med
  - Lettere at vedligeholde ved større projekter
  - Programlogik og HTML relativt adskilt
  - Forskellige personer kan tage sig af programmering og HTML-design
  - Centraliseret
    - Programlogik og beslutning om, hvilken side, der skal vises sker ét sted

Model 1 og 2 webarkitektur

- Mange bud på implementation af model 2
  - To slags JSP-sider: Nogen har kun programlogik, andre kun HTML
  - JSP og javabønner
  - "Model-View-Controller"
    - Frontkontrol-servlet/JSP fortolker og behandler inddata
    - dirigerer videre til præsentations-JSP der genererer HTML
  - Struts: Overbygning der giver hændelseshåndtering
    - Struts agerer Frontkontrol ud fra XML-fil
  - Java Server Faces: Visuel designer a la grafiske applikationer
    - JSF giver hændelseshåndtering og generer bagvedliggende kode
  - JSP-sider, der genererer XML,
    - HTML-designer skriver XSLT (XML-transformationer) til HTML
  - Hjælpeteknologier
    - Servlet-filtre (præprocessering og omdirigering af forespørgsler)
    - Taglibs: HTML-lignende koder der udføres på serveren
      - JSTL
      - Kan simplificere JSP-siderne betydeligt

JSP og Javabønner

En *javabønne* er en klasse som opfylder to specielle krav:

- Klassen har en konstruktør uden parametre.
- Data i bønnen kan manipuleres med get- og set-metoder. Sådanne metoder kaldes også *egenskaber*.

Eksempel: En Person-bønne med egenskaben *fornavn*:

```
package minPakke;

public class Person
{
  private String navnet;          // intern variabel (ikke egenskab)

  public Person()                 // kontruktør uden parametre
  {
  }

  public String getFornavn()
  {
    return navnet;
  }

  public void setFornavn(String n)
  {
    navnet = n;
  }
}
```

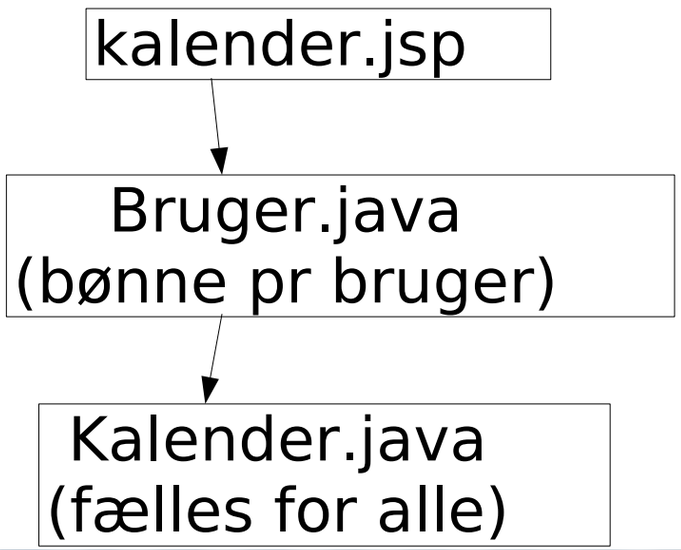
De indsendte data fra formularen der passer med bønnens egenskaber laves om til set-kald til bønnen:

```
<html>
<head>
<title>simpel bønne i JSP-side</title>
</head>
<body>

<jsp:useBean id="person" scope="session" class="minPakke.Person" />
<jsp:setProperty name="person" property="*" />

<% if (person.getFornavn() == null) { %>
  Indtast dit navn:
  <form>
  <input type="text" name="fornavn">
  <input type="submit" value="OK">
  </form>
<% } else { %>
  Hej <%= person.getFornavn() %>!
<% } %>
</body>
</html>
```

JSP og Javabønner - eksempel

Fælleskalender for oktober 2001
forrige måned - næste måned - rediger kalenderen.

03 on JA, så er kalenderen klar til brug, god fornøjelse - Jacob og Anne.
04 to kl. xx-xx, torsdagsgruppe/ Pia
05 fr åbent 11-16/Evy+Anne, menú : hakkebøf m. kartofler, spinat m.m
06 lø fri
07 sø
08 ma
09 ti åbent 11-16/ Frank+Yvonne, menú :
10 on
11 to
12 fr
13 lø
14 sø
15 ma
16 ti åbent 11-16/Frank+Yvonne, menú :




```
<html><head><title>kalender</title></head>
<body bgcolor="#03D5E4">
<%-- Knyt bønnen Bruger til brugerens session under navnet b --%>
<jsp:useBean id="b" scope="session" class="kalender.Bruger" />

<%-- Overfør parametre til b, der svarer til egenskaber --%>
<jsp:setProperty name="b" property="*" />

<h1>Fælleskalender for <%= b.getDatostr() %></h1>
<a href="kalender.jsp?maaned=<%= b.getMaaned()-1 %>">forrige</a> måned -
<a href="kalender.jsp?maaned=<%= b.getMaaned()+1 %>">næste</a> måned -
<%
  if (b.isRediger()) { // redigering - vis en input-formular
%>
  <a href="kalender.jsp?rediger=false">vis</a> kalenderen.<br />
  <br />
  <form action="kalender.jsp" method="post">
    Tryk <input type="submit" value="OK"> når du vil gemme ændringer.<br />
>

    <% b.udskrivDagsprogram(out); %>

    <br /><br />
    <input type="submit" value="OK">
  </form>
<%
  } else {              // fremvisning
%>
  <a href="kalender.jsp?rediger=true">redigér</a> kalenderen.<br />
  <% b.udskrivDagsprogram(out); %>
<%
  }
```


```
package kalender;
import java.util.*;  import java.text.*;  import java.io.*;

public class Bruger
{
  static Locale dk = new Locale("da","DK");
  static SimpleDateFormat månedFormat = new SimpleDateFormat("MMMM yyyy",dk);
  static SimpleDateFormat dagugedagFormat = new SimpleDateFormat("dd EE",dk);

  private boolean redigering;
  public void setRediger(boolean r) { redigering = r; }
  public boolean isRediger() { return redigering; }

  private GregorianCalendar dato = new GregorianCalendar(dk);

  public void setMaaned(int m) {
    dato.set(Calendar.MONTH, m);
    dato.set(Calendar.DAY_OF_MONTH, 1); // første dag, så hele måneden ses
  }
  public int getMaaned() { return dato.get(Calendar.MONTH); }

  /** Giver aktuelle måned og år som en streng */
  public String getDatostr() { return månedFormat.format(dato.getTime()); }

  /** Egenkaben dagsprogram er et array af strenge, en for hver dag.
   *  der kaldes videre i det fælles Kalender-objekt                  */
  public void setDagsprogram(String[] dagsprogram) {
    redigering = false;
    int start = Kalender.instans.beregnIndex(dato);
    for (int i=0; i<dagsprogram.length; i++)
      Kalender.instans.sæt(start+i,dagsprogram[i]);
  }

  /** Udskriv et dagsprogram */
  public void udskrivDagsprogram(Writer out) throws IOException {
    GregorianCalendar kal = (GregorianCalendar) dato.clone();
    int start = Kalender.instans.beregnIndex(kal);
    int antal = 1 + dato.getActualMaximum(Calendar.DAY_OF_MONTH)
                  - dato.get(Calendar.DAY_OF_MONTH);
    for (int i=0; i<antal; i++) {
      String dagugedag = dagugedagFormat.format(kal.getTime());
      out.write("<br /><code>");
      out.write(dagugedag);
      out.write("</code> ");
      if (!redigering) out.write(Kalender.instans.hent(i+start));
                  + Kalender.instans.hent(i+start).replace('\'','\"') + "'>");
```


```
package kalender;
import java.util.*; import java.text.*; import java.io.*;

public class Kalender
{
  private boolean ændret;
  private List liste;
  /**
   * @param kal Datoen vi ønsker at kende indekset i listen på
   * @returns indekset der skal bruges i kald til sæt() og hent().
   */
  public int beregnIndex(Calendar kal) {
    int år = kal.get(Calendar.YEAR);
    int dag = kal.get(Calendar.DAY_OF_YEAR);
    // det vigtigste er at to dage aldrig får samme indeks
    return (år-2001)*366+dag;
  }

  /**
   * Sæt teksten for en bestemt dag. Indeks
   * @see #beregnIndeks(Calendar)
   * @param indeks Indekset i listen. Skal først findes med beregnIndeks()
   * @param tekst Teksten for dagen
   */
  public void sæt(int indeks, String tekst) {
    // Fyld op med tomme strenge hvis der gås ud over listen
    while (indeks>=liste.size()) liste.add("");
    liste.set(indeks,tekst);
    ændret = true;
  }

  /**
   * Hent teksten for en bestemt dag.
   * @see #beregnIndeks(GregorianCalendar)
   * @param index Indekset i listen. Skal først findes med beregnIndeks()
   * @return tekst Teksten for dagen
   */
  public String hent(int indeks) {
    if (indeks<0 || liste.size()<=indeks) return "";
    else return (String) liste.get(indeks);
  }

  public static final Kalender instans = new Kalender(); // singleton med

  private Kalender() {                                    // privat konstruktør
    try { // indlæs kalenderen fra disken (hvis den findes)
```

Åben Dokumentlicens

- Dette foredragsmateriale er under Åben Dokumentlicens (ÅDL)
  - Du har derfor lov til frit at kopiere dette værk
  - Bruger du dele af værket i et nyt værk, skal de dele, der stammer fra dette værk, igen frigives under ÅDL
  - Den fulde licens kan ses på http://www.sslug.dk/linuxbog/licens.html