



# Java-opgraderingskursus



## Danmarks Meteorologiske Institut Gang 2

- Webprogrammering
  - Klient-server og forespørgsel/svar
  - HTML og HTML-formularer
  - Servleter og JSP-sider
- Fælles biblioteker
  - Programmere i pakker
  - Lave JAR-fil
  - Dokumentation
- Læse og skrive filer
  - Formatering og fortolkning af datoer og tal
  - Formatering og fortolkning af en datafil
    - Udskrive i kolonner
    - Regulære udtryk og andre måder at opdele en streng i bidder

Dette materiale er under Åben Dokumentlicens, se <http://www.sslug.dk/linuxbog/licens.html>



## HTML: Hypertekst Markup Language



- HTML: Tekst + HTML-koder (i < og >)
- HTML-koder instruerer fremviseren i hvordan tekst skal vises

```
<html>
<head><title>Simpel hjemmeside</title></head>
<body>

<h1>En simpel hjemmeside</h1>

<p>Velkommen til min lille <i>hjemmeside</i>.
</p>

<p>Jeg hedder <b>Jacob</b> og underviser på
<a href="http://www.cv.ihk.dk">
Center for Videreuddannelse</a> på
<a href="http://ihk.dk">
Ingeniørhøjskolen i København</a>.
</p>

<p>Her kan du se hvordan jeg ser ud:<br>
</p>

</body>
</html>
```





# Java Server Pages



- HTML-sider med Java-kode i
  - Java-koden fortolkes og udføres på serveren
  - Oversættes til binær (maskin)kode ved første forespørgsel

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

<%
out.println( "<h1>Hej verden!</h1>" );
out.println( "To plus to er: " );
out.println( 2 + 2 );
%>

</body>
</html>
```

## Klienten modtager:

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

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

</body>
</html>
```



# 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
  - Response-objektet bruges til at skrive HTML-kode i.

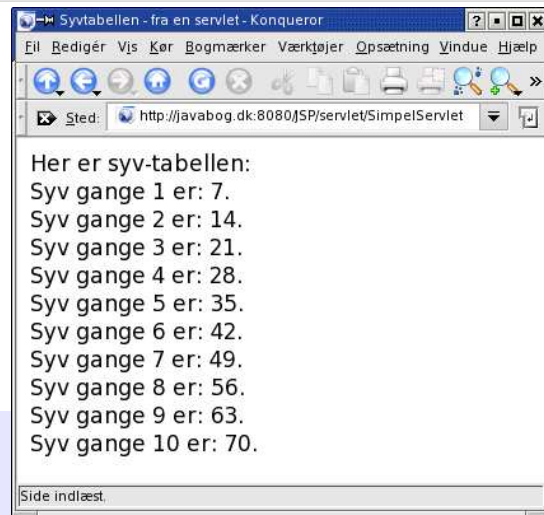
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HejServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws IOException
    {
        response.setContentType("text/html");
        PrintWriter ud = 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

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SempelServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Syvtabellen - fra en servlet</title></head>");
        out.println("<body>");
        out.println("<p>Her er syv-tabellen:<br>");

        for (int i=1; i<=10; i++)
        {
            out.println("Syv gange "+ i +" er: "+ 7*i +"<br>");
        }
        out.println("</body>");
        out.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>SempelServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>En simpel servlet</servlet-name>
  <url-pattern>/servlet/SempelServlet</url-pattern>
</servlet-mapping>
...
</web-app>
```



# request-objektet



```
<html>
<head><title>Data om klienten</title></head>
<body>
<h1>Nogle data om klienten (request-objektet)</h1>
<pre>
URL          - request.getRequestURL(): <%= request.getRequestURL() %>

Metode       - getMethod():          <%= request.getMethod() %>
Protokol     - getProtocol():        <%= request.getProtocol() %>
Værtsnavn   - getServerName():      <%= request.getServerName() %>
Port        - getServerPort():       <%= request.getServerPort() %>
URI         - getRequestURI():       <%= request.getRequestURI() %>

Klients IP-adresse - getRemoteAddr(): <%= request.getRemoteAddr() %>
Klients maskinnavn - getRemoteHost(): <%= request.getRemoteHost() %>
Foretrukne sprog - getLocale():      <%= request.getLocale() %>
Netlæser/browser header user-agent: <%= request.getHeader("user-agent") %>
</pre>
</body>
</html>
```

```
URL          - request.getRequestURL(): http://javabog.dk:8080/JSP/kode/kapitel_02/data

Metode       - getMethod():          GET
Protokol     - getProtocol():        HTTP/1.1
Værtsnavn   - getServerName():      javabog.dk
Port        - getServerPort():       8080
URI         - getRequestURI():       /JSP/kode/kapitel_02/data_om_klienten.jsp

Klients IP-adresse - getRemoteAddr(): 80.62.163.37
Klients maskinnavn - getRemoteHost(): 80.62.163.37
Foretrukne sprog - getLocale():      da
Netlæser/browser header user-agent: Mozilla/5.0 (compatible; Konqueror/3.1; Linux
```



# Formularer og interaktive sider



```
<html>
<head><title>formular</title></head>
<body>
<h3>Hvad hedder du?</h3>
<form>
<input type="text" name="navn">
</form>
</body>
</html>
```



```
<%
String navnet = request.getParameter("navn");
```



# Formularer og interaktive sider



- 1) Klienten laver en forespørgsel
- 2) Serveren sender en HTML-side med formular
- 3) Klienten viser HTML-side med formular
- 4) Bruger udfylder formular og trykker 'OK'
- 5) Klienten laver en ny forespørgsel med formularens data
- 6) Serveren fortolker formulardata
- 7) Serveren sender en ny HTML-side



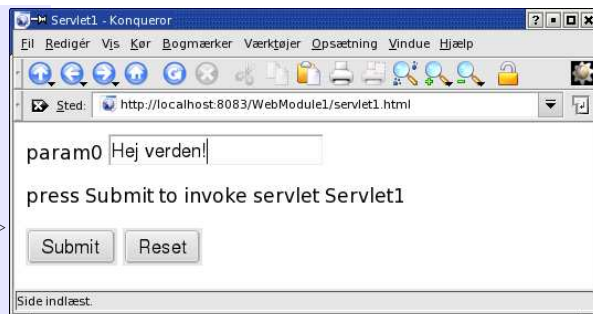
# Formularer og interaktive sider



```
<html>
<head>
<title>
Servlet1
</title>
</head>
<body bgcolor="#ffffff">

<form action="/WebModule1/servlet1" method="get">
<p>param0 <input type="text" name="param0"></p>
<p>press Submit to invoke servlet Servlet1</p>
<p><input type="submit" name="Submit" value="Submit">
<input type="reset" value="Reset"></p>
</form>
</body>
</html>
```

```
public class Servlet1 extends HttpServlet
{
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
    {
        String var0 = request.getParameter("param0");
        if (var0 == null) var0 = "";
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Servlet1</title></head>");
        out.println("<body bgcolor=\"\#ffffff\">");
        out.println("<p>The servlet has received a GET. This is the reply.</p>");
        out.println("<p>Du skrev: "+var0+"</p>");
        out.println("</body>");
        out.println("</html>");
        out.close();
    }
}
```





# Formularer og interaktive sider



Elementer i en formular - Konqueror

Eil Redigér Vjs Kør Bogmærker Værktøjer

Sted: elementer\_i\_en\_formular.html

## Elementer i en formu

Skriv dit navn (tekstfelt):

og din kode (kodefelt):

Beskriv dig selv (tekstområde):

Hvad foretrækker du at programmere i (radioknapper):  C  C++  Java

Hvad kan du programmere i (afkrydsningsfelter):  C  C++  Java

Hvilken ret foretrækker du (valgliste):

Hvilke retter kan du spise (valgliste):

```

<p>Skriv dit navn (tekstfelt):
<input type="text" name="navn" value="et navn" size="10" />
<br />og din kode (kodefelt):
<input type="password" name="kode" value="hemli" size="10" />
<input type="hidden" name="id" value="1234">
</p>

<p>
Beskriv dig selv (tekstområde): <br />
<textarea name="beskrivelse" rows="2" cols="30">Jeg taler espe
</textarea>
</p>

<p>Hvad foretrækker du at programmere i<br />(radioknapper):
<input type="radio" name="foretr_prg" value="c"/>C
<input type="radio" name="foretr_prg" value="cpp"/>C++
<input type="radio" name="foretr_prg" value="java" checked="che
</p>

<p>Hvad kan du programmere i<br />(afkrydsningsfelter):
<input type="checkbox" name="kan_prg" value="c" checked="che
<input type="checkbox" name="kan_prg" value="cpp" />C++
<input type="checkbox" name="kan_prg" value="java" checked="che
</p>

<p>Hvilken ret foretrækker du (valgliste): <br />
<select name="foretr_spise">
<option selected="selected">Spaghetti med kødsovs</option>
<option>Pizza</option>
<option>Ostefondue</option>
</select>
</p>

<p>Hvilke retter kan du spise (valgliste): <br />

```



# Eksempel: Login



```

<html>
<head><title>login1</title></head>
<body>
<h1>Log ind</h1>

<form method="post" action="login2.jsp">
Brugernavn:<input type="text" name="brugernavn" /><br />
Adgangskode:<input type="password" name="adgangskode" /><br />
<input type="submit" value="Log ind" />
</form>
<p>
Vink: Brugernavnet er "Jacob" og adgangskoden er "hemli".
</p>
</body>
</html>

```

login1 - Konqueror

Eil Redigér Vjs Kør Bogmærker Værktøjer Opsætning Vindue Hjælp

Sted: http://localhost:8080/kapitel\_05/login1.html

## Log ind

Brugernavn:

Adgangskode:

Vink: Brugernavnet er "Jacob" og adgangskoden er "hemli".

Færdig.



# Eksempel: Login



```
<html>
<head><title>login2</title></head>
<body>

<%
// hvis brugernavn="Jacob" og adgangskode="hemli" logges der ind.
// dette burde selvfølgelig hentes fra en database eller lign.
if ("Jacob".equals(request.getParameter("brugernavn"))) &&
    "hemli".equals(request.getParameter("adgangskode")))
{
    // sæt attributten "logget ind" i sessionen
    session.setAttribute("logget ind", "ja");
    out.println("Du er logget korrekt ind.");
}
else
{
    // fjern attributten "logget ind" fra sessionen
    session.removeAttribute("logget ind");
    out.println("Forkert brugernavn eller adgangskode.");
}
%>
```

Du kan nu prøve at gå videre til `<a href="login3.jsp">den beskyttede side</a>` eller gå tilbage og logge ind igen.

```
</body>
</html>
```



# Eksempel: Login



```
<%
// se om attributten "logget ind" er sat i sessionen
if (session.getAttribute("logget ind") == null) {
    // brugeren er ikke logget ind, så send ham tilbage til login-siden
    response.sendRedirect("login1.html");
}
%>
```

```
<html>
<head><title>login3</title></head>
<body>
```

```
<h1>Den beskyttede side</h1>
Denne tekst kan du kun se, hvis du er logget korrekt på.
```

```
</body>
</html>
```







## Mere om JSP: Sessioner



- Hver bruger får tildelt et session-objekt når de besøger en JSP-side.
- Sessionen følger brugeren, lige meget hvilken side han/hun er inde på, og er derfor nyttigt til at huske data, der skal følge brugeren.



## Mere om JSP: Sessioner



```
<h3>Skriv et ønske</h3>
Skriv noget, du ønsker.
<form>
<input type="text" name="oenske">
</form>
<%
// hent listen over ønsker
ArrayList liste = (ArrayList) session.getAttribute("oensker");

if (liste == null) { // hvis
    liste = new ArrayList(); // opret
    session.setAttribute("oensker", liste); // og sæt
}

// se om der kommer en parameter med endnu et ønske
String ønske = request.getParameter("oenske");
if (ønske != null) {
    liste.add(ønske); // tilføj
}

if (liste.size()>0) {
    %>
    <h3>Ønskeseddel</h3>
    Indtil nu har du følgende ønsker:<br />
    <%
    // udskriv hele listen
    for (int i=0; i<liste.size(); i++)
    { %>
        Ønske nr. <%= i %>: <%= liste.get(i) %><br />
    }
    %>
}
%>
```

Dette eksempel demonstrerer, hvordan session-objektet kan bringe huske brugerens indtastninger.

**Skriv et ønske**

Skriv noget, du ønsker.

En million i Lotto

**ønskeseddel**

Indtil nu har du følgende ønsker:

ønske nr. 0: En ny bil  
ønske nr. 1: Et stort hus

Færdig.





# Implicit definerede objekter



- Der findes en række implicit definerede objekter, som man altid har adgang til i en JSP-side:
  - **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 model 2 i webserverprogrammering



## Historisk perspektivering

### Før: Model 1 - programlogik sammen med HTML

- Simpel struktur
- Nem at starte med
- Velegnet til små projekter
- Svært (umuligt) at adskille programlogik og HTML
- Samme person er programmør og HTML-designer
- Potentiel redundans (samme programlogik flere steder)

### Nu: Model 2 - programlogik adskilt fra HTML

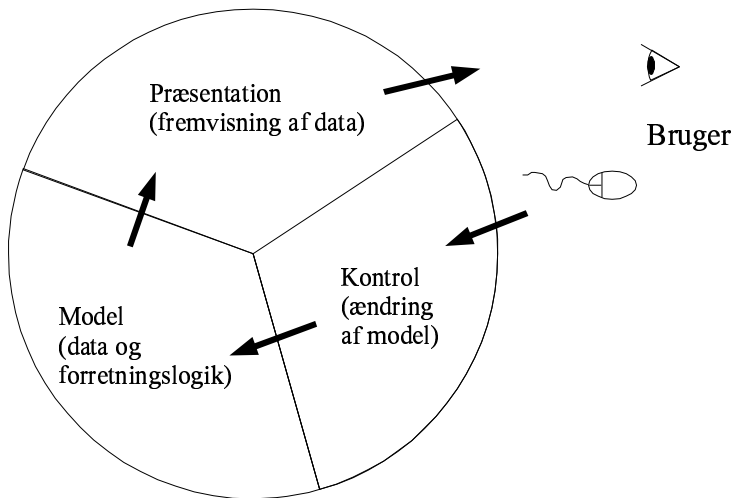
- Adskil tekstligt indhold og programkode fra hinanden, sådan at f.eks. en HTML-designer kan koncentrere sig om HTML-layout og indhold, mens en programmør koncentrerer sig om funktionaliteten og den bagvedliggende kode.
- Mere omfattende struktur
  - Sværere at starte med
  - Lettere at vedligeholde ved større projekter
  - Programlogik og HTML relativt adskilt
  - Forskellige personer kan tage sig af programmering og HTML-design
  - Programlogik ét sted

Mange bud på implementation af model 2 (kaldet model 2a, 2b, 2c, ...):

- To slags JSP-sider: Nogen har kun programlogik, andre kun HTML
- JSP og javabønner
- "MVC" (kontrol-servlet/JSP fortolker og behandler inddata og dirigerer videre til præsentations-JSP)
- Taglibs: HTML-lignende koder der udføres på serveren
- Struts: Overbygning der giver hændeshåndtering a la grafiske applikationer
- Programmør skriver JSP-sider, der genererer XML, HTML-designer skriver XSLT (XML-transformationer) til HTML
- Servlet-filtre, ...



# Model-View-Controller



## Platforms- og serveruafhængighed



- Java er platformsuafhængigt
  - kører på Linux, Mac, Windows, Unix, ...
  - JSP-webapplikation er derfor platformsuafhængig
- Standarder for struktur på webapplikation
  - Konfiguration f.eks. altid i WEB-INF/web.xml
  - Standard er del af J2EE-specifikationen
    - Tomcat er referenceimplementation
  - Mange alternativer
    - Tomcat, Oracle (OC4J), BEA Weblogic, Sun ONE, Resin, ...

Resultat: Frit valg af platform og server



# Installation af en webapplikation



- WAR-fil (Web ARchive)
  - = ZIP-fil med hel applikation
- Hent WAR-fil
- Læg i webapps/ og den installeres automatisk
- Al opsætning findes i WEB-INF/web.xml



# Fælles biblioteker



- Programmere i pakker
- Lave JAR-fil
- Dokumentation



# Programmere i pakker



- En pakke: En samling af relaterede klasser
- En klasse svarer til en fil på filsystemet
- En pakke svarer til et underkatalog på filsystemet

```
// Filnavn: src/minPakke/Klassel.java
package minPakke;
import java.util.*;

public class Klassel
{
    public void snak()
    {
        System.out.println("Dette er Klassel, der taler!");
    }
}
```

```
// Filnavn: src/BenytPakker.java
import minPakke.*;
import java.util.*;

public class BenytPakker
{
    public static void main(String[] arg)
    {
        Klassel a = new Klassel();
        a.snak();
    }
}
```



# Lave JAR-fil



- JAR-fil er en ZIP-fil med klasser
  - jar cf program.jar BenytPakker.class minPakke
  - zip -r program.jar BenytPakker.class minPakke
- Værktøjet kan lave den for en!



# Dokumentation



## Javadoc

```
/**
 * Eksempel på en kommenteret klasse.
 */
public class EnKommenteretKlasse
{
    /**
     * Et eksempel på en metode. Metoden tjener
     * til at vise hvordan javadoc virker.
     *
     * @param enStreng strengen
     * @param etTal tallet
     *
     * @return strengen og tallet sat sammen
     */
    public String enMetode(String enStreng,
                          int etTal)
    {
        return enStreng+etTal;
    }
}
```

javadoc EnKommenteretKlasse.java

De vigtigste klasser bør være dokumenteret med Javadoc!

### Class EnKommenteretKlasse

```
java.lang.Object
|
|--EnKommenteretKlasse
```

```
public class EnKommenteretKlasse
extends java.lang.Object
```

Eksempel på en kommenteret klasse.

#### Constructor Summary

[EnKommenteretKlasse\(\)](#)

#### Method Summary

```
java.lang.String enMetode(java.lang.String enStreng, int etTal)
Et eksempel på en metode.
```

#### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

#### Constructor Detail

#### EnKommenteretKlasse

```
public EnKommenteretKlasse()
```

#### Method Detail

#### enMetode

```
public java.lang.String enMetode(java.lang.String enStreng,
int etTal)
```

Et eksempel på en metode. Metoden tjener til at vise hvordan javadoc virker.

#### Parameters:

enStreng - strengen  
etTal - tallet

#### Returns:

strengen og tallet sat sammen



# Formatering og fortolkning af datoer og tal



```
package dmi;

import DMI.VU.util.DateTime;

public class TalOgDatoer
{
    public static void main(String[] args)
    {
        DateTime t = new DateTime();
        System.out.println(t.getDateTimeString("dd-MM-yy"));
        System.out.println(t.getDateTimeString("dd,MM,yy"));

        System.out.println(t.getDateTimeString("dd/MM HH:mm"));
    }
}

10-11-04
10,11,04
10/11 07:11
```



# Formatering og fortolkning af en datafil



- Udskrive i kolonner
- Regulære udtryk og andre måder at opdele en streng i bidder