javabog.dk − Webprogrammering med Java Server Pages − Forord

Indholdsfortegnelse

Indholdsfortegnelse

Indholdsfortegnelse

Indholdsfortegnelse

Indholdsfortegnelse

Indholdsfortegnelse

0 Forord

Denne bog henvender sig til de, der gerne vil lære at lave webprogrammering med JSP − Java Server Pages. Det er en fordel hvis du allerede har kendskab til Java på et vist niveau.

0.1 Bogens opbygning

Kapitel 1, Introduktion, introducerer HTML og grundprincipperne i serverside−teknologi og viser en række udviklingsværktøj og webservere der understøtter JSP.

Kapitel 2, Grundelementer i JSP, introducerer JSP−syntaks og de grundlæggende begreber.

Kapitel 3, Interaktive sider, beskriver hvordan brugeren får mulighed for at sende data til serveren og interaktionen mellem HTML−koden, der typisk indeholder en HTML−formular, netlæserens fremvisning af denne og parametre der sendes til serveren.

Kapitel 4, Videre med JSP, introducerer sessions−begrebet og omdirigering og hvordan det kan bruges til at lave en interaktiv webapplikation.

Kapitel 5, Brug af databaser, introducerer kort databaser, hvordan man arbejder med dem og hvordan man kommunikerer med dem fra JSP. De mange forskellige strategier til databaseadgang beskrives også.

Kapitel 6, JSTL − JSP Standard Tag Library, er frivillig læsning, der ikke forudsættes læst i resten af bogen. Det beskriver JSTL, et HTML−lignende sprog, som kan bruges i stedet for Java, når man skriver koden, der udføres på serveren. Er du ikke så erfaren med Java, kan du vælge at lære JSTL i stedet, ved at kigge på og lege med eksemplerne i dette kapitel.

Kapitel 7, Inde i webserveren, er også frivillig læsning. Det introducerer servletter og hvordan JSP−sider internt bliver oversat til servletter. Derefter ses begrebet webapplikation, hvordan man laver en, hvordan den pakkes og et samlet eksempel på web.xml vises.

Kapitel 8, Sikkerhed og adgangskontrol, der også er frivillig læsning, beskriver kryptering med SSL og hvordan man kan få webserveren til beskytte visse sider med brugernavn og adgangskode. Sidst diskuteres sikkerhed og hvordan man beskytter sig mod hacking.

Kapitel 9, Javabønner i JSP−sider, viser hvordan man med fordel benytter javaklasser til at huske data og til større opgaver, der er uhensigtsmæssige at udføre direkte i JSP−siden, bl.a. oprettelse og identifikation af brugere og sende elektronisk post.

Kapitel 10, Arkitekturer i webprogrammering, diskuterer hvad der sker, når en webapplikation vokser sig stor og designmønstre til at håndtere dette, herunder trelagsmodellen, model 1− og 2−arkitekturer, Model−View−Controller−arkitekturen og Frontkontrol.

Kapitel 11, XML, indholdssyndikering og webtjenester, introducerer kort XML og relaterede teknologier og beskriver forskellige anvendelser, bl.a. indholdssyndikering. Derudover introduceres metodekald over netværket og webtjenester.

Kapitel 12, Enterprise JavaBeans, handler om store serversystemer og J2EE−platformen, med fokus på Enterprise JavaBeans.

Kapitel 13, Internationale sider, beskriver hvordan man får en webapplikation til at fungere internationalt, d.v.s. på flere nationale sprog.

Forrest i hvert kapitel beskrives forudsætningerne for at læse kapitlet, så her kan du vurdere, om der er et andet kapitel, du eventuelt bør se i først.

0.1.1 Hvert kapitels opbygning

- Hvert kapitel starter med en **oversigt** over indholdet, og hvilke andre kapitler du forudsættes at have læst, så du altid har overblik over, om du har de nødvendige forudsætninger.
- Dernæst kommer **hovedteksten**, der introducerer emnerne og kommer med eksempler hvor de anvendes.

De fleste kapitler har herefter

- **Appendiks** giver en komplet oversigt over de fakta, som relaterer sig til kapitlet.
  Det er beregnet til at blive læst sammen med kapitlet, men er også velegnet til senere opslag.
- **Test dig selv**, hvor du kan afprøve, om du har fået fat i de vigtigste ting i kapitlet.
- **Resumé**, der i kort form repeterer de vigtigste ting.
- **Avanceret**, der handler om de mere avancerede ting i relation til kapitlets emne.
  Disse kan springes over ved første læsning, men kan være nyttige at kigge på senere hen. Resten af bogen forudsætter ikke, at man har læst de avancerede afsnit.

0.1.2 Til underviseren

Bogen udspringer af noter fra kurset "Videregående Programmering", som jeg begyndte at undervise i i efteråret 2001 på IT−Diplomuddannelsen på Ingeniørhøjskolen i København. Den er i høj grad resultatet af praktisk undervisning og derfor velegnet til kursusbrug.

Som underviser kan du i starten af hvert kapitel se, hvad det forudsætter og planlægge læsningen gennem kurset derefter

Der findes en (gratis) samlet undervisningspakke, hvor underviseren får:

- transparenter, der supplerer bogen
- forslag til semesterplan og opgavesedler
- vejledende opgavebesvarelser

Undervisningspakken kan hentes på http://javabog.dk/JSP. Du er også meget velkommen til at skrive til mig, på nordfalk@mobilixnet.dk, hvis du holder et kursus og er interesseret i supplerende stof eller har spørgsmål til undervisningspakken.

0.2 Netudgaven af bogen

Størstedelen af bogen findes også som netudgave, på adressen http://javabog.dk/JSP.

På http://javabog.dk/JSP kan du gratis læse ca. 80% af bogen og derudover hente:

- Programeksemplerne fra bogen
- Alt koden kan gennemses på http://javabog.dk/JSP/kode/.
- Koden kan afprøves (udført på en server) på: http://javabog.dk:**8080**/JSP/kode/.
- Du kan også hente netudgaven af hele bogen med alle eksempler som en WAR−fil (webapplikation, klar til at installere og køre i din egen JSP−webserver, se hvordan i afsnit 7.4, Webapplikationer).
- Undervisningsmateriale fra et kursus i Web− og serverprogrammering, såsom lektionsplaner, ugesedler, opgaver og transparenter

- Foredrag om udvalgte kapitler (som lydfiler)

- Rettelser og tilføjelser til bogen

Netudgaven, d.v.s. alt der kan findes på http://javabog.dk/JSP, er frigivet under Åben Dokumentlicens (ÅDL). Det betyder, at du kan udskrive og kopiere netudgaven, som du lyster. Den præcise ordlyd af Åben Dokumentlicens kan læses på næste side.

I netudgaven er ca. 20% af teksten skåret væk, men bogen er stadig fuldt ud brugbar, idet kun undværlige emner er udeladt, såsom avancerede afsnit og hjemmeopgaver.

Bidrag og bemærkninger er meget velkomne. Send dem til nordfalk@mobilixnet.dk.

0.3 Tak

Tak til Mikala Kjellerup, Kim Damgaard, Jakob Raavig, Tina Modvig Frederiksen, Troels Nordfalk og Henrik Tange for deres kritik og forslag til forbedringer.

Tak til Sune Cording for at lade mig bruge hans tekst som udgangspunktet for afsnit 11.5, Webtjenester, og til René Thomsen for idéen til afsnit 2.8.6, Indlejre og nedskalere billeder fra harddisken.

Særligt mange tak til min allerkæreste Anne Mette for støtte og tålmodighed og for at være kommet med mange gode kommentarer til kapitel 11, XML, indholdssyndikering og webtjenester. Og tak til min søn Aske for at være så glad og nem som spæd at jeg bl.a. har kunnet gøre denne bog færdig (i skrivende stund er han 3 måneder).

Sidst men ikke mindst: Tak til Linux−samfundet for at lave et styresystem, der styrer! Denne bog er skrevet med OpenOffice.org under Mandrake Linux. Begge har åben kildekode (Open Source) og kan hentes på http://OpenOffice.org og http://mandrakelinux.com gratis.

Dankon por la granda helpo, kiu ebligis tiun i libron!
(det er esperanto og betyder "tak for den store hjælp, som gjorde denne bog mulig!")

Jacob Nordfalk

Valby, september 2004.

0.4 Åben Dokumentlicens (ÅDL)

Version 1.0 (25. januar 2002)

1. Kopiering og distribution af værket i uændret form

Du må frit kopiere dette værk uændret og distribuere det på ethvert medium, forudsat at du sammen med hver kopi bibeholder denne licens og en henvisning til værkets kilde.

Du kan vælge at kræve penge for kopiering og distribution af indholdet, for undervisning i indholdet, for brugerstøtte og garantier i forbindelse med indholdet. Hvis du videregiver værket til andre, har du pligt til at se til, at du selv eller andre giver modtageren gratis adgang i minimum 3 år til en elektronisk udgave af indholdet − i et åbent redigérbart format − for eksempel via internettet, og at der i hvert eksemplar tydeligt angives, hvordan man kan få den elektroniske udgave.

2. Ændring af indholdet

Du må gerne ændre i dette værk eller dele af det (og dermed lave et nyt værk baseret på dette værk) og distribuere det som nævnt ovenfor, forudsat at du opfylder alle følgende krav:

a. Du skal sørge for, at det nye værk indeholder en tydelig anmærkning om, at du ændrede det, hvad du ændrede og datoen for ændringerne.
b. Det skal være omfattet af samme rettigheder og betingelser ("licens") som det oprindelige værk.

Alternativt kan det nye værk bære en anden licens eller almindelig ophavsretlig beskyttelse, forudsat at samtlige dele, der oprindeligt stammer fra dette værk, tydeligt er markeret (f.eks. i sidefoden på hver side) som værende under denne licens, og at licensen følges for disse dele af værket, herunder at der gives gratis elektronisk adgang som nævnt under afsnittet om kopiering og distribution.

3. Pligter

Enhver anvendelse, kopiering eller distribuering betragtes som en accept af denne licens og dens betingelser. Ved overtrædelse af licensbetingelserne, vil almindelig ophavsretlig beskyttelse være gældende for hele værket og vil dermed gøre person eller personer erstatningspligtige for enhver uautoriseret anvendelse.

4. Garanti

Indholdet leveres "som det er", uden nogen form for garanti for købs−, brugs− eller nytteværdi, medmindre der er givet en særskilt skriftlig garanti herom.

Denne bog er under ÅDL, med undtagelse af avancerede afsnit, Test dig selv, Resumé og løsninger på opgaver samt kapitel 8, Sikkerhed og adgangskontrol, kapitel 12, Enterprise JavaBeans. Disse dele er under almindelig ophavsretlig beskyttelse.

javabog.dk | << forrige | indhold | næste >> | programeksempler | om bogen

---

1 Introduktion

En overfladisk forståelse af dette kapitel forudsættes i det meste af bogen.

Java Server Pages (forkortet JSP) giver dig en lang række af muligheder for at gøre din hjemmeside mere levende.

JSP benyttes i mange sammenhænge, hvor man ønsker dynamiske indhold i HTML−sider, bl.a. i debatfora, postlister, afstemninger og e−handels−løsninger.

Målet med denne bog er, at give en grundig indførsel i, hvordan du kommer i gang med at bruge JSP. Den starter fra bunden, men kræver dog, at du i forvejen har et godt kendskab til HTML. JSP er mere avanceret end HTML og det er en stor fordel hvis du allerede har prøvet at programmere i Java (eller et sprog med samme syntaks, f.eks. JavaScript eller C/C++), da man kan bruge Java som programmeringssprog i JSP. Databasekendskab er også en fordel, da man ofte har brug for at gemme nogle oplysninger.

Hvis du undervejs får brug for hjælp kan du besøge nogle af henvisningerne i afsnit 1.6. Der finder du svar på spørgsmål lige fra helt banale begynderproblemer til de meget avancerede spørgsmål.

1.1 Hvad er HTML?

Hjemmesider er skrevet i sproget HTML (HyperText Markup Language). Det forudsættes at du har rimeligt kendskab til HTML i forvejen, så hvis nedenstående kommer som en overraskelse for dig, bør du nok starte med en HTML−vejledning i stedet.

Her er et eksempel på en HTML−side:


```
<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>
<img src="jacob.jpg"></p>

</body>
</html>
```

Til højre ses hvordan en netlæser (eng.: web browser), her Konqueror under Linux, viser siden. Det, der ikke er indkapslet i < og > er tekst, der bliver vist på skærmen.

Koderne mellem < og > kaldes HTML−koder (eng.: tags) og fortæller netlæseren hvordan den skal vise teksten. Eksempelvis markerer <h1> at der kommer en overskrift, der skal vises med større skrift end resten.

De fleste koder skal afsluttes med en slut−kode (f.eks skal <h1> afsluttes med </h1>). Koderne kan have attributter, der giver ekstra information (f.eks har koden <img>, der bruges til at vise et billede, attributten src, der angiver filnavnet på billedet, der skal vises det pågældende sted: <img src="jacob.jpg">).

1.2 Hvad er JSP?

En JSP−side er en en HTML−side med endelsen ***.jsp***, som udover HTML også indeholder noget programkode, der er skrevet i Java. Javakoden afvikles på webserveren, og resultatet sendes til klienten, dvs. brugerens netlæser (eng.: web browser).

JSP (Java Server Pages) er udviklet af Sun Microsystems og fællesskabet omkring Java.

1.2.1 Princippet i serverside−teknologier

Den bedste måde at forklare, hvordan en serverside−teknologi (og dermed JSP) fungerer på er at sammenligne det med almindelig HTML.

En HTML−side

Forestil dig at du i din netlæsers adresselinje taster adressen (URL[1]) på et HTML−dokument, f.eks. http://minesider.dk/dokument**.html**. Det kaldes at *anmode* (eng.: request) om en HTML−side. Da sker der det følgende:

- Serveren finder HTML−filen
- Serveren sender filen til klienten

Serveren sender altså bare HTML−filen afsted til klienten.

En JSP−side

Hvis du i stedet taster en URL på en JSP−side, f.eks. http://minesider.dk/dokument**.jsp**, og dermed anmoder om et JSP−dokument, kommer serveren på arbejde:

- Serveren finder JSP−filen
- Serveren kigger filen igennem for Java−kode
- Serveren udfører de beregninger, Java−koden foreskriver
- Serveren sender resultatet til klienten

Det er vigtigt at forstå at klienten kun ser *resultatet* af serverens beregninger − ikke selve instrukserne. JSP stiller derfor ikke krav til hvilken netlæser dine besøgende har, men til serveren som dine sider ligger på.

Hvis du vælger "vis kilde" (eng.: View Source) når du anmoder om at se et JSP−dokument, kan du ikke se JSP−koderne − kun almindeligt HTML. Du kan altså ikke se, hvordan en JSP−side er lavet, men er nødt til at lære JSP på andre måder − f.eks. ved at læse denne bog.

Denne bog handler om at skrive kode, der udføres på serveren, og det betyder at du skal installere et program på din computer, som gør den i stand til at fungere som en server. Der er naturligvis også den mulighed, at du bruger en eksisterende server, der understøtter JSP(f.eks. et webhotel), og at du så lægger dine sider der, når du vil prøve dem.

1.3 Udviklingsværktøjer til JSP

Det antages at du allerede har et redigeringsværktøj til HTML og ved hvordan det bruges.

Desuden har du brug for adgang til en PC eller en webserver, som kan afvikle JSP.

Den mest skrabede løsning, man kan vælge at redigere JSP−siderne i, er et ikke−Java−orienteret program, som for eksempel Notesblok under Windows eller kedit (eller emacs) under Linux. Så er det nødvendigt at installere en server, som kan afvikle JSP (se afsnit 1.4).

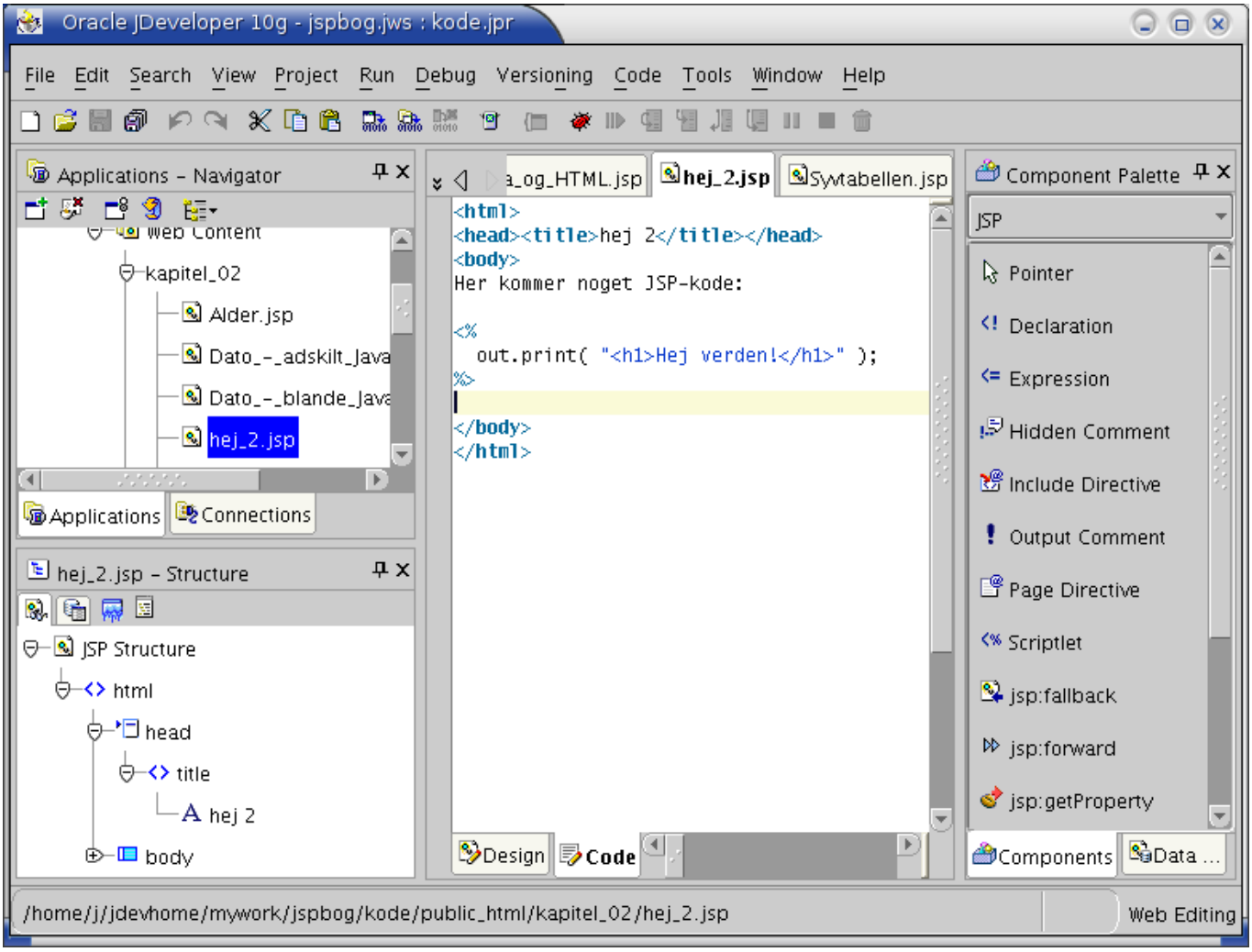
Når man programmerer synes de fleste dog, at det er rarere at benytte et Java−udviklingsmiljø, der kan hjælpe en med:

- Redigering af kildetekster (skrevet i HTML, JSP og Java).
- Oversættelse af kildeteksten og tjek for om den er syntaktisk korrekt.
- Nem kørsel og fejlfinding (i udviklingsmiljøets indbyggede webserver).

De fleste udviklingværktøjer er opbygget med en menulinje øverst, der indeholder tilgang til filhåndtering, projektstyring og alle nødvendige værktøjer, hvoraf den vigtigste er "Run". "Run" oversætter først kildeteksten og starter derefter fortolkeren. Uddata kan ses i den nederste ramme. "Debug" (der findes under "Run") bruges til fejlfinding af programmer og giver mulighed for at udføre programkoden trinvist.

1.3.1 Oracle JDeveloper

Databaseproducenten Oracle udgiver udviklingsværktøjet JDeveloper:



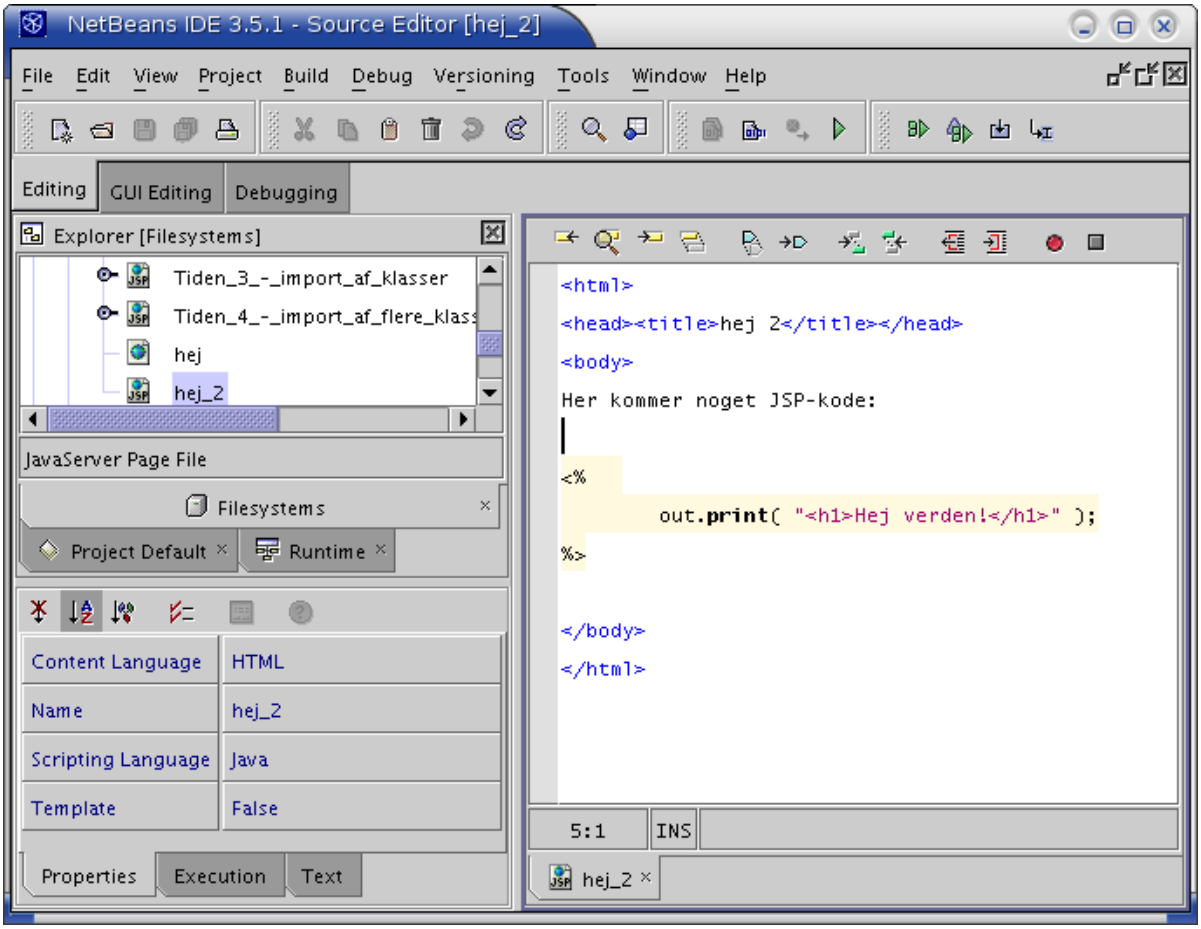
JDeveloper kan en masse meget avancerede ting, men kan for begyndere være indviklet at bruge, på grund af de mange muligheder.

JDeveloper er skrevet i Java, kræver 256 MB RAM og fås bl.a. til Windows og Linux.

Den fulde udgave kan hentes gratis til privat- og undervisningsbrug på http://oracle.com. Til kommercielt brug skal man købe en licens.

1.3.2 Netbeans og Sun Java Studio Creator

Sun har udgivet sit eget udviklingsmiljø, Sun Java Studio Creator (tidligere kaldet Forte og Sun ONE Studio), også skrevet i Java, til Windows, Linux og Sun Solaris.



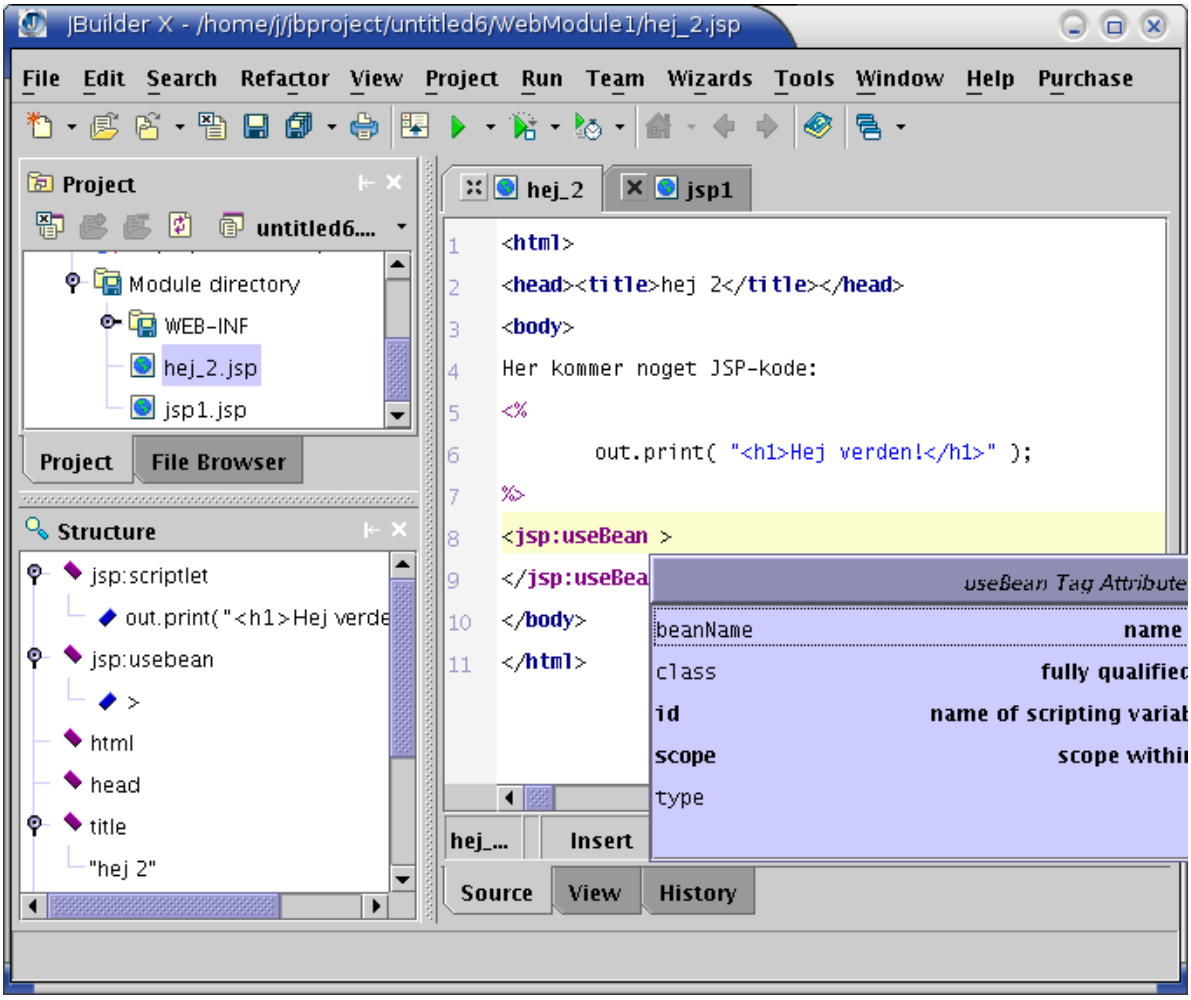
Udviklingsværktøjet understøtter udvikling af JSP (sørg for at have alle web−moduler installeret):

Vælg først File|New...|JSP & Servlets|Web Module og vælg en mappe du vil have dine JSP−filer, og derefter File|New...|JSP & Servlets|JSP og vælg mappen du lige har oprettet og giv et filnavn til JSP−siden (uden .jsp−endelse, det gør værktøjet selv). Højreklik på filen og vælg 'Execute' for at køre JSP−filen i værktøjets interne webserver.

En basisudgave af programmet har fået Åben Kildekode (Open Source) under navnet NetBeans og kan hentes gratis på http://netbeans.org. En prøveudgave af Sun Java Studio Creator kan hentes gratis på http://developers.sun.com/prodtech/javatools/jscreator/

1.3.3 Borland JBuilder

JBuilder fra Borland er et af de mest populære Java−udviklingsmiljøer. JBuilder er skrevet i Java og kører på både Macintosh, Windows, Linux og Sun Solaris. Det anbefales at have 256 MB RAM.



En basisversion af JBuilder, der desværre ikke tillader udvikling i JSP, kan hentes gratis fra http://borland.com/jbuilder. Ønsker man adgang til de mere avancerede funktioner, herunder udvikling i JSP, skal programmet købes eller man må nøjes med en prøveversion, der kun virker i 30 dage.

1.3.4 Eclipse og IBM WebSphere Studio

IBM WebSphere Studio − baseret på Open Source−initiativet Eclipse (http://eclipse.org) er et andet stærkt udviklingsmiljø til bl.a. JSP. En gratis prøveversion kan hentes på IBMs hjemmeside: http://www−306.ibm.com/software/awdtools/studioappdev/.

1.4 Installation af en server

De fleste udviklingsværktøjer understøtter udvikling af JSP−sider og har en server indbygget.

> Har du et Java−udviklingsmiljø der understøtter webprogrammering behøver du *ikke* installere en server for at prøve at programmere JSP−sider

Er du begynder foretrækker du sandsynligvis at lade udviklingsmiljøet tage sig af detaljerne og bruge dens server.

Vil du installere din egen server skal du have et Java−udviklingskit (kaldet JDK eller J2SE) installeret på din maskine. Java kan hentes fra http://java.sun.com, men har du allerede et Java−relateret udviklingsmiljø (f.eks JBuilder, JDeveloper, Netbeans/Forte, ...) på din maskine, er Java allerede installeret[2].

Er du i tvivl kan det sjældent skade at installere Java en gang til :−)

Der findes rigtig mange Java−webservere på markedet, lige fra helt små (endda en, der kun fylder 2 megabyte) til kæmpeprogrammer fra store firmaer som Borland, Oracle og IBM:

- Apache Tomcat − http://jakarta.apache.org/tomcat/
- Resin − http://www.caucho.com/
- IBM Websphere − http://www−4.ibm.com/software/webservers/
- Oracle Application Server − http://www.oracle.com/appserver/java_edition.html
- BEA WebLogic − http://www.bea.com/products/weblogic/
- Borland Enterprise AppServer − http://www.borland.com/besappserver/
- Macromedia JRun − http://www.macromedia.com/software/jrun/
- Orion Application Server − http://www.orionserver.com/
- Jetty − http://jetty.mortbay.com/jetty/
- Blazix Java Application Server − http://www.blazix.com/

Kun de første to vil blive beskrevet i det følgende.

1.4.1 Tomcat

Jakarta Tomcat (http://jakarta.apache.org/tomcat/) er Suns officielle implementering af en JSP−server. Den har for programudviklere en speciel status som reference−implementering, sådan at den, i tvivlstilfælde, definerer hvordan en JSP−webserver bør fungere, og viser dermed de andre producenter af JSP−servere, hvad de har at rette sig efter.

Tomcat har Åben Kildekode (eng.: Open Source) og er gratis at bruge både til ikke−kommercielle og kommercielle formål.

Den kan hentes på adressen http://jakarta.apache.org/builds/jakarta−tomcat/. Hjemmesiden er lidt svær at finde rundt i, men du skal ende med at stå med en fil der hedder f.eks. jakarta−tomcat−5.0.27.exe (installationsprogram − anbefales hvis du bruger Windows) eller jakarta−tomcat−5.0.27.zip (kan bruges under både Linux, Windows og andre styresystemer).

Versionsnummeret kan selvfølgelig variere, men til de ting du kommer til at bruge her gør det ikke den store forskel hvilken version du bruger.

Bruger du ZIP−filen er der et par skridt du skal ordne selv:

Installation af ZIP−filen − Linux

Pak ZIP−filen med Tomcat ud, f.eks. i din hjemmemappe og gå ind i bin−mappen:

```
cd jakarta-tomcat-5.0.27/bin
```

Miljøvariablen JAVA_HOME skal være sat til at pege på din Java−installation.

Det kan du gøre ved at skrive (erstat stien med der, hvor Java er installeret)

```
export JAVA_HOME=/usr/local/jdk1.4
```

på kommandolinjen eller ved at redigere filen catalina.sh eller setclasspath.sh og indsætte linjen øverst. Sørg også for at alt i bin−mappen kan eksekveres (chmod a+x *).

Start derefter Tomcat:

```
./startup.sh
```

Installation af ZIP−filen − Windows

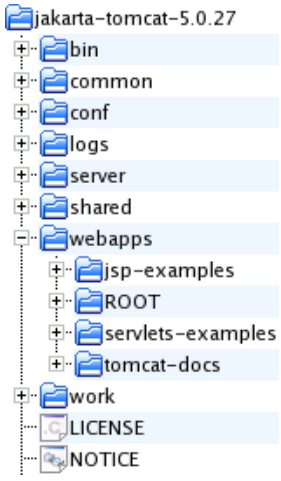
Pak ZIP−filen med Tomcat ud, f.eks. i C:\ (roden på C−drevet). Derefter skal du ind i kontrolpanelet og definere miljøvariablen JAVA_HOME til at pege på der, hvor JDK er installeret (f.eks. C:\JBuilderX\jdk1.4). Du kan også gå ind i Tomcats bin−mappe og redigere filen catalina.bat eller setclasspath.bat og sætte linjen

```
set JAVA_HOME=C:\JBuilderX\jdk1.4
```

ind øverst i filen (erstat stien med der, hvor Java er installeret). Det kan være du også skal højreklikke på startup.bat og under Egenskaber vælge at reservere mere hukommelse til miljøvariabler.

Start derefter Tomcat, ved at dobbeltklikke på startup.bat.

Prøve installationen



Du kan nu gå i gang med at lege med Tomcat og prøve JSP−eksemplerne, der følger med Tomcat. Det er også en god idé at udforske filstrukturen (med f.eks. Stifinder) og blive fortrolig med mappestrukturen (vist til højre).

Tjek først at Tomcat kører ved at åbne http://localhost:8080/. Gør den ikke det, må du læse installationsvejledningen der følger med Tomcat nøjere og evt. kigge på log−filerne (der ligger i undermappen logs/).

Websiderne er opdelt i grupper, der kaldes webapplikationer. Hver webapplikation ligger under dens eget navn i en mappe under webapps/.

F.eks. ligger JSP−eksemplerne under webapps/**jsp−examples**/ på filsystemet og kan tilsvarende findes i gennem webserveren på adressen http://localhost:8080/**jsp−examples**/.

Webapplikationer forklares nærmere i afsnit 7.4.

Hvis du har problemer med at se JSP−sider

Hvis JSP−eksemplerne ikke virker (men du i øvrigt godt kan se almindelige HTML−sider og servletter gennem din Tomcat på http://localhost:8080/) så er det nok fordi Tomcat ikke kan finde filen tools.jar (der findes i din Java−installation, sandsynligvis under jdk1.4/lib/tools.jar). Så får du nok fejlen "java.lang.NoClassDefFoundError: sun/tools/javac/Main", når du forsøger at se en JSP−side.

Sørg da for, at Tomcat kender jdk1.4/lib/tools.jar fra JDKet, f.eks. ved at indføje den i CLASSPATH eller kopiere den ind i ext/−mappen for din JDK (jdk1.4/jre/lib/ext/). Se evt. afsnit 4.9.6, Hvis klasse(bibliotek)er ikke kan findes.

Bemærk at du skal have JDK (Java Developers Kit, også kaldet J2SE − Java 2 Standard Edition) installeret. JRE (Java Runtime Environment) er ikke tilstrækkeligt.

1.4.2 Resin

En anden populær webserver er Resin fra Caucho (http://caucho.com). Den er der mange der synes er betydeligt lettere at installere og bruge end Tomcat.

Resin skal blot hentes fra http://caucho.com og pakkes ud, hvorefter httpd.exe (Windows) eller httpd.sh (Linux), der ligger inde i bin−mappen, startes.

Derefter kan man prøve den ved at åbne http://localhost:8080/, klikke på Demo og prøve JSP−eksemplerne (de ligger som filer i doc/examples/basic/).

Resin har Åben Kildekode og er gratis at bruge til ikke−kommercielle formål. Til kommercielle formål skal en licens købes.

Den rareste ting ved Resin i forhold til Tomcat viser sig først, når man begynder at arbejde med servletter og separate java−klasser: Resin oversætter selv automatisk .java−kildetekstfilerne til binære .class−filer og indlæser dem. I Tomcat skal man selv sørge for at oversætte .java−kildetekst til .class−filer.

1.5 PHP og ASP − JSPs fætre

JSP er, som vi vil se i næste kapitel, HTML−sider med noget ekstra programkode, der udføres på serveren. Dette princip er også brugt i en række andre web−programmeringssprog, såsom det ekstremt populære PHP (der står for PHP: Hypertext Preprocessor) og det mindre populære ASP (Active Server Pages) fra Microsoft.

1.5.1 JSP fremfor PHP og ASP − fordele og ulemper

- Kender man allerede Java er JSP det nemmeste at gå til.
- Kender man ikke Java er JSP sværere at lære end PHP eller ASP. JSP bruger de almindelige Java−klasser og følger Java−syntaksen, som bl.a. skelner mellem store og små bogstaver, og det kan være svært for begyndere.
  Man kan dog anvende JSTL i sine JSP−sider i stedet. JSTL (der er beskrevet i afsnit 2.1.5) er et HTML−lignende sprog, der er betydeligt lettere at lære end Java.
- JSP (og Java) er platformsuafhængigt, så man kan frit vælge om ens server skal køre på Windows, Linux, Mac eller andre UNIX'er (PHP kører også næsten alle platforme, mens ASP i praksis kun kører på Microsoft Windows).
- Webapplikationer skrevet i JSP kan køre på en række forskellige webservere som f.eks. Apache Tomcat, BEA WebLogic, IBM WebSphere, Oracle Application Server, Resin og en snes andre. Populære webservere som Apache og IIS (Microsoft Internet Information Server) kan nemt udvides til at understøtte JSP via et plugin.
- JSP er ikke så udbredt til små projekter som PHP eller ASP. Du vil sandsynligvis finde flere på nettet, der kan hjælpe dig med disse sprog end med JSP.
- Der er også flere webhoteller, der understøtter PHP eller ASP end JSP. Hvis du planlægger at bruge et webhotel (i stedet for at installere din egen server) skal du lige tjekke, om du kan finde et der passer til dine behov og understøtter JSP.
- JSP er meget udbredt til større projekter, som det er meget velegnet til.
- JSP er en velintegreret del af J2EE (Java 2 Enterprise Edition), som er et meget udbredt serversystem der findes bred understøttelse for i industrien.

Som tommelfingerregel er det en god idé at vælge JSP hvis du kender Java i forvejen eller hvis du gerne vil lære et sprog og en metodik, der også fungerer sammen med store serversystemer.

1.6 Videre læsning

- http://html.dk har en glimrende introduktion til HTML og relaterede sprog som CSS (typografier i hjemmesiden − eng: Cascading Style Sheets) og JavaScript (kode der køres i klientens netlæser).

Man kan læse mere om JSP på de følgende sider:

- http://java.sun.com/products/jsp/technical.html

- http://jsptut.com

Har du brug for hjælp til JSP eller installation af serverne så kig forbi

- Diskussionsgruppen dk.edb.programmering.java
  (kan findes på http://groups.google.dk/groups?q=dk.edb.programmering.java)
- http://www.eksperten.dk
- På http://java.sun.com/products/jsp/jsp−asp.html kan du se en sammenligning mellem JSP og ASP.

1 URL − Uniform Resource Locator − den adresse, som en hjemmeside har på internettet.

2I det tilfælde kan du bruge JDKet der følger med udviklingsmiljøet. I de fleste ligger JDKet i en mappe med navnet JDK, f.eks. ligger Borlans JBuilders JDK i mappen JBuilderX/jdk1.4/)

2 Grundelementer i JSP

En grundig forståelse af emnerne i dette kapitel forudsættes i resten af bogen.
Bemærk: De avancerede emner i slutningen af kapitlet forudsættes ikke, de er frivillig læsning, du senere kan vende tilbage til.

I dette kapitel laver vi den allerførste helt simple JSP−fil. Her kan du også teste, om din server er sat rigtigt op og kan køre JSP.

Grundlæggende er en JSP−fil en tekstfil med endelsen ***.jsp***, som består af:

- Tekst
- HTML−koder
- JSP−programkode

Tekst og HTML−koder kender du, så lad os kigge lidt mere på JSP−programkoden.

2.1 JSP−programkode

Når man programmerer JSP så skriver man i virkeligheden i programmeringssproget Java. Derfor kan det være en god ide at lære lidt Java undervejs, f.eks. ved at kigge på de første kapitler af af en lærebog som http://javabog.dk.

2.1.1 En simpel side med HTML−kode

Start med at lave et ganske almindeligt HTML−dokument, navngiv filen *hej.jsp*, og placer den sammen med de andre JSP−filer der fulgte med din webserver (med Tomcat er stien nok jakarta−tomcat−5.0.27/webapps/ROOT/hej.jsp). HTML−koden skal se sådan her ud:

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

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

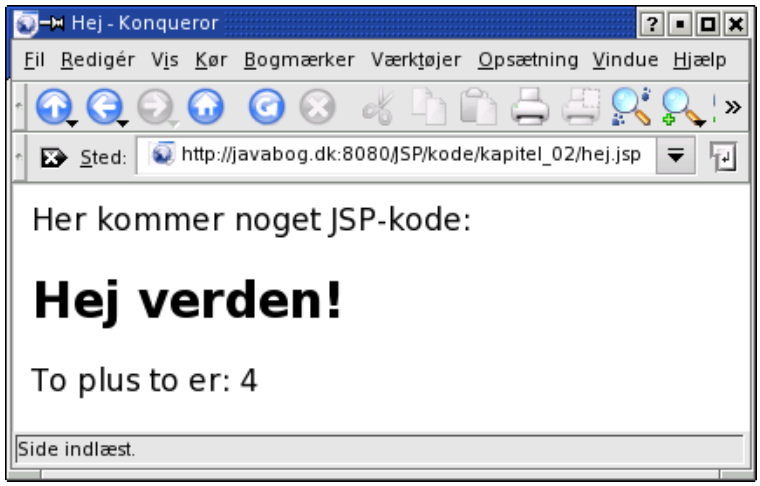
Prøv så at hente siden gennem webserveren.

Sandsynligvis er adressen http://localhost:8080/hej.jsp.

2.1.2 Lægge JSP−serverkode ind i siden

Husk at JSP er at skrive opgaver til en server, og lad os så prøve at skrive en opgave til serveren. For at fortælle serveren, hvad den skal kigge efter, må vi have nogle koder som fortæller, hvornår serverkoderne *starter* og *slutter*. I JSP bruger man **<%** og **%>** til at markere start og slut på det, som serveren skal udføre.

Prøv derefter at tilføje følgende helt simple kodestump til din HTML−kode:




```
<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>
```

Når du nu ser JSP−dokumentet i en netlæser, skulle resultatet gerne være som vist til højre.

Men det interessante viser sig først, når du får vist HTML−koden i netlæseren (ved at vælge "vis kilde" eller "View Source"):

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

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

</body>
</html>
```

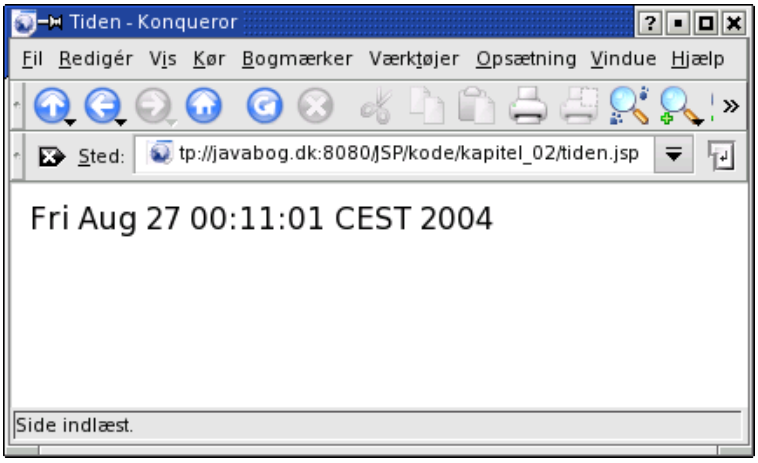
JSP−koderne er væk. Som allerede nævnt er det kun serveren som kan se JSP−koderne − klienten (netlæseren) ser kun resultatet af beregningerne. Serveren har udført Java−koden:

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

Lidt teknisk udtrykt ville man sige, at her brugte vi out−objektets metode println() til at skrive strenge til klienten. Bemærk at Java−kommandoer altid afsluttes med semikolon (;).

2.1.3 Eksempel: Dags dato og tid

Lad os prøve at få serveren til at skrive noget andet. Vi kunne f.eks. bede den om at udskrive den aktuelle dato og tidspunkt:




```
<html>
<head><title>Tiden</title></head>
<body>
<%
  out.print( new java.util.Date() );
%>
</body>
</html>
```

Vi kan altså få serveren til at udskrive dato og tidspunkt, når JSP−siden vises. Bemærk, at hvis man trykker på opdater/genindlæs/reload i sin netlæser, vises et nyt tidspunkt. Serveren beregner og udskriver altså dato og tidspunkt hver gang siden sendes til en klient.

Det er også væsentligt at bemærke, at den HTML−kode klienten ser, ikke indeholder andet end datoen. Det vil sige at JSP ikke stiller krav til, hvilken netlæser, der anvendes. Det er således sådan, at ting, der er lavet med serverside teknologier, virker i *alle* netlæsere.

2.1.4 Hvis du ikke har en server til rådighed

Hvis du ikke har en server, der kan køre JSP, til rådighed, kan du prøve alle eksemplerne fra bogen på http://javabog.dk:8080/JSP/kode/. Ovenstående eksempel har således adressen http://javabog.dk:8080/JSP/kode/kapitel_02/hej.jsp .

2.1.5 JSTL − en ny måde at arbejde på

Er du ikke så erfaren med Java er JSTL (JSP Standard Tag Library), der beskrives i kapitel 6, JSTL − JSP Standard Tag Library, nok lettere for dig end Java. JSTL er et HTML−lignende sprog som man kan skrive koden, der udføres på serveren i, i stedet for Java.

I kapitel 6 er der en række eksempler med JSTL, der gør det samme som eksemplerne i dette og de efterfølgende kapitler. Selvom denne bogs 'hovedsprog' er Java og ikke JSTL kan du derfor vælge at lære JSTL i stedet ved at læse hovedteksten og så ved Java−eksemplerne i stedet kigge på og lege med JSTL−eksemplerne i kapitel 6.

2.2 Variabler

En variabel kan opfattes som en navngiven papirlap, hvor der til enhver tid kan stå netop én ting. Variabler bruges til at huske data.

Variabler i Java skal altid erklæres, dvs. at man skal fortælle serveren, at der skal oprettes en variabel, hvad slags data den skal indeholde og hvad den skal hedde:

```
  int alder = 31;
```

Her har vi oprettet variablen alder. Den er af typen int, det vil sige at den kan indeholde hele tal. Samtidig har vi givet (eller tildelt) den værdien 31.

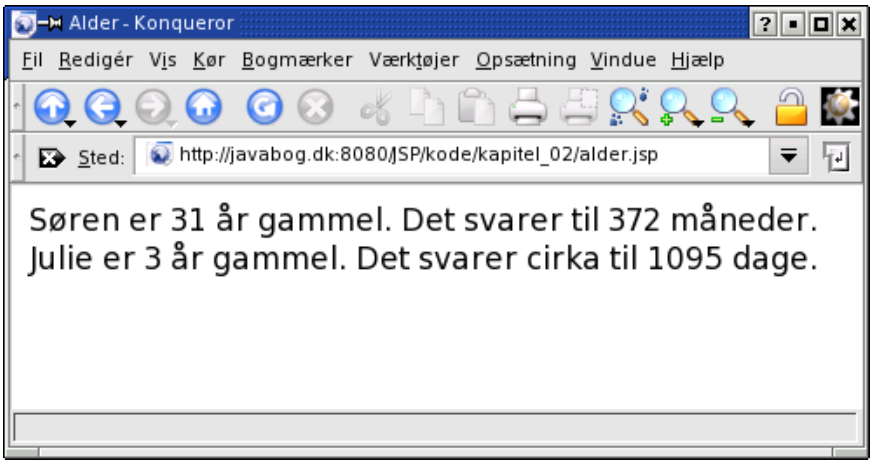
Senere kunne vi så hente værdien frem ved at skrive variablens navn. Her er et eksempel:

alder.jsp (JSTL−eksempel i afsnit 6.1)

```
<html>
<head><title>Alder</title></head>
<body>
<p>
<%
  int alder = 31;

  out.print( "Søren er " + alder + " år gammel. " );
  out.print( "Det svarer til " + 12*alder + " måneder. <br>" );

  alder = 3;
```

```
  out.print( "Julie er " + alder + " år gammel. " );

  alder = alder*365;
  out.print( "Det svarer cirka til " + alder + " dage. <br>" );
%>
</p>
</body>
</html>
```

Resultatet i en netlæser er:



Bemærk, at når en variabel tildeles en ny værdi, bliver den gamle værdi fuldstændig glemt.

Bemærk også, at på højre side af lighedstegnet i en tildeling kan stå et regneudtryk, der i så fald beregnes før variablen tildeles værdien.

Udtrykket alder = alder*365 gør altså det, at det beregner alder*365, der giver 3*365=1095. Denne værdi puttes derefter ind i variablen alder.

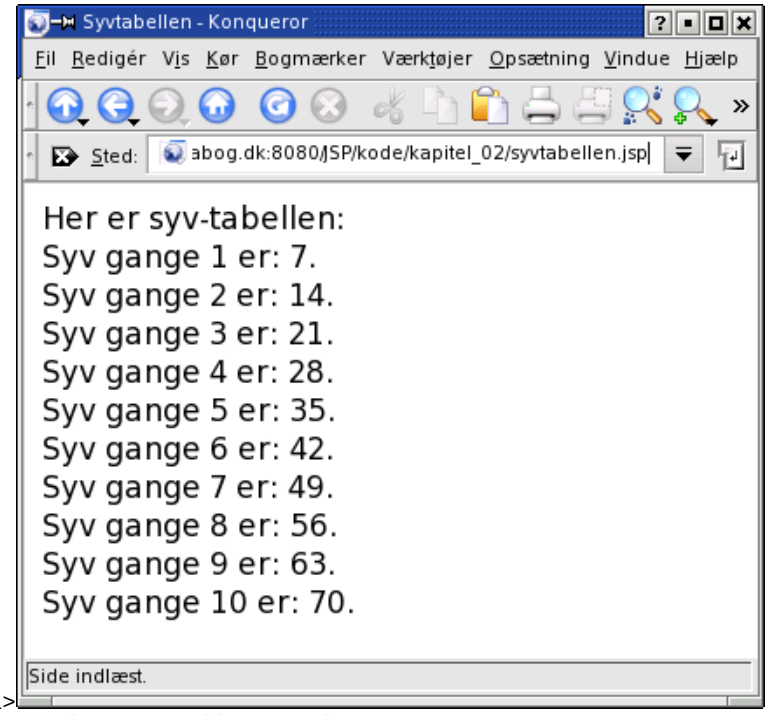
2.2.1 Indlejrede java−udtryk

I koden <%= %> kan skrives et Java−udtryk, der bliver beregnet på serveren og indsat i HTML−koden hver gang siden hentes.

<%= dato %> er altså bare en kortere måde at skrive <% out.print(dato); %> på.

Her er en side, der udskriver 7−tabellen ved hjælp af <%= %>:

syvtabellen.jsp (JSTL−eksempel i afsnit 6.1.5)




```
<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>
```

Bemærk at en klump javakode gerne må slutte, selvom der mangler en }−parentes til at afslutte den blok kode, der skulle udføres i for−løkken, blot der senere kommer endnu en klump javakode, hvori blokken afsluttes (det kommer vi tilbage til i afsnit 2.3.1).

2.2.2 Variabler med objekter

I eksemplet i afsnit 2.1.3 skrev vi **new java.util.Date()**, hvilket opretter et objekt, som repræsenterer den aktuelle dato og tid på serveren.

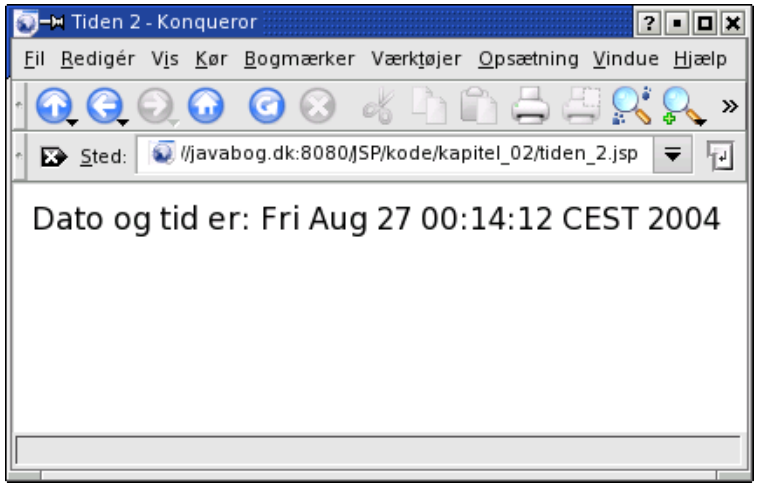
Sådan et objekt kan man også gemme i en variabel, sådan her:

```
  java.util.Date tiden = new java.util.Date();
```

og så bagefter udskrive objektet på skærmen:

```
  out.print( tiden );
```

Lad os udvide eksemplet og både udskrive en streng og et objekt, som vi samler med +:




```
<html>
<head><title>Tiden 2</title></head>
<body>
<%
  java.util.Date tiden = new java.util.Date();
  out.print( "Dato og tid er: " + tiden );
%>
</body>
</html>
```

Når netlæseren anmoder om siden, udfører serveren JSP−koden og sender til klienten:

```
<html>
<head><title>Tiden 2</title></head>
<body>
Dato og tid er: Wed Aug 28 16:00:42 CEST 2002
</body>
</html>
```

2.2.3 Importere klassedefinitioner (pakker)

I stedet for hele tiden at skrive Date−klassens fulde navn (som er java.util.Date) kan vi også importere java.util.* (alle klasser i pakken java.util) og således undgå at skrive pakkenavnet.

Til det skal vi bruge side−direktivet <%@ page ... %>, der kan bruges til at angive en række ting om hvordan serveren skal udføre siden (side−direktivet bliver grundigere beskrevet senere, i afsnit 4.4):

```
<%@ page language="java" import="java.util.*" %>
<html>
<head><title>Tiden 3 − import af pakke</title></head>
<body>
<%
  Date tiden = new Date();
  out.print( "<p>Dato og tid er: " + tiden + "</p>" );
```

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

Importere flere pakker

Vil man importere klasser fra flere pakker skriver man dem adskilt af komma, f.eks. import="java.util.*,java.text.*".

Med det kan vi lave en lidt pænere side, der udskriver datoen og tiden:

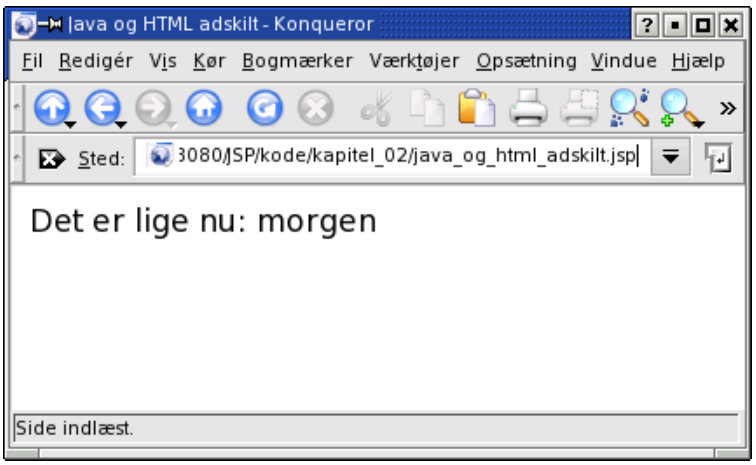
```
<%@ page language="java" import="java.util.*,java.text.*" %>
<html>
<head><title>Tiden 4 - import af flere pakker</title></head>
<body>
<%
    DateFormat klformat, datoformat;

    klformat   = DateFormat.getTimeInstance(DateFormat.SHORT);
    datoformat = DateFormat.getDateInstance(DateFormat.FULL);

    Date tid = new Date();
    String kl = klformat.format(tid);
    String dato = datoformat.format(tid);
%>
<p>Datoen i dag er <%= dato %>, og klokken er <%= kl %>.</p>
</body>
</html>
```

Resultatet i en netlæser er:

2.3 Blanding af HTML og Java − to stilarter

Lad os skrive tidspunktet på dagen. Det kan vi gøre med en række kædede if−sætninger:

java_og_html_adskilt.jsp (JSTL−eksempel i afsnit 6.2.2)




```
<%@ page language="java" import="java.util.*" %>
<html>
<head>
<title>Java og HTML adskilt</title></head>
<body>
Det er lige nu:
  <%
    Date tid = new Date();
    int t = tid.getHours();
    if (t <= 9)       out.print("morgen");
    else if (t <= 12) out.print("formiddag");
```

```
    else if (t <= 17) out.print("eftermiddag");
    else if (t <= 21) out.print("aften");
    else              out.print("nat");
  %>
</body>
</html>
```

De fleste foretrækker at bruge out−objektet til udskrivning, som vist ovenfor, for på denne måde at adskille Java og HTML.

2.3.1 Blandet Java og HTML

Men det er faktisk muligt at starte og stoppe javakoden med <% og %> i stedet:

```
<%@ page language="java" import="java.util.*" %>
<html>
<head><title>Java og HTML blandet</title></head>
<body>
Det er lige nu:
  <%
    Date tid = new Date();
    int t = tid.getHours();
    if (t <= 9)       { %> morgen      <% }
    else if (t <= 12) { %> formiddag   <% }
    else if (t <= 17) { %> eftermiddag <% }
    else if (t <= 21) { %> aften       <% }
    else              { %> nat         <% }
  %>
</body>
</html>
```

For de fleste er det forvirrende med denne sammenblanding af HTML og Java. Derfor foretrækker de fleste at bruge out−objektet (en tredje mulighed er at bruge JSTL, se hvordan i afsnit 6.2).

Dette eksempel kan dog virke besnærende for nogen, fordi det ser mere overskueligt ud. Man skal dog være meget omhyggelig med at omkranse HTML−koden i {− og }−parenteser.

> Husk altid at bruge { og } i betingelser og løkker hvis de har indlejret HTML
>
> Man skal være omhyggelig med, at der altid er lige mange {−startparenteser som }−slutparenteser

Er der ikke lige mange {− og }−blokparenteser får man kryptiske[1] fejlmeddelelser såsom:

```
  'try' without 'catch' or 'finally'
  'catch' without 'try'
  'else' without 'if'
```

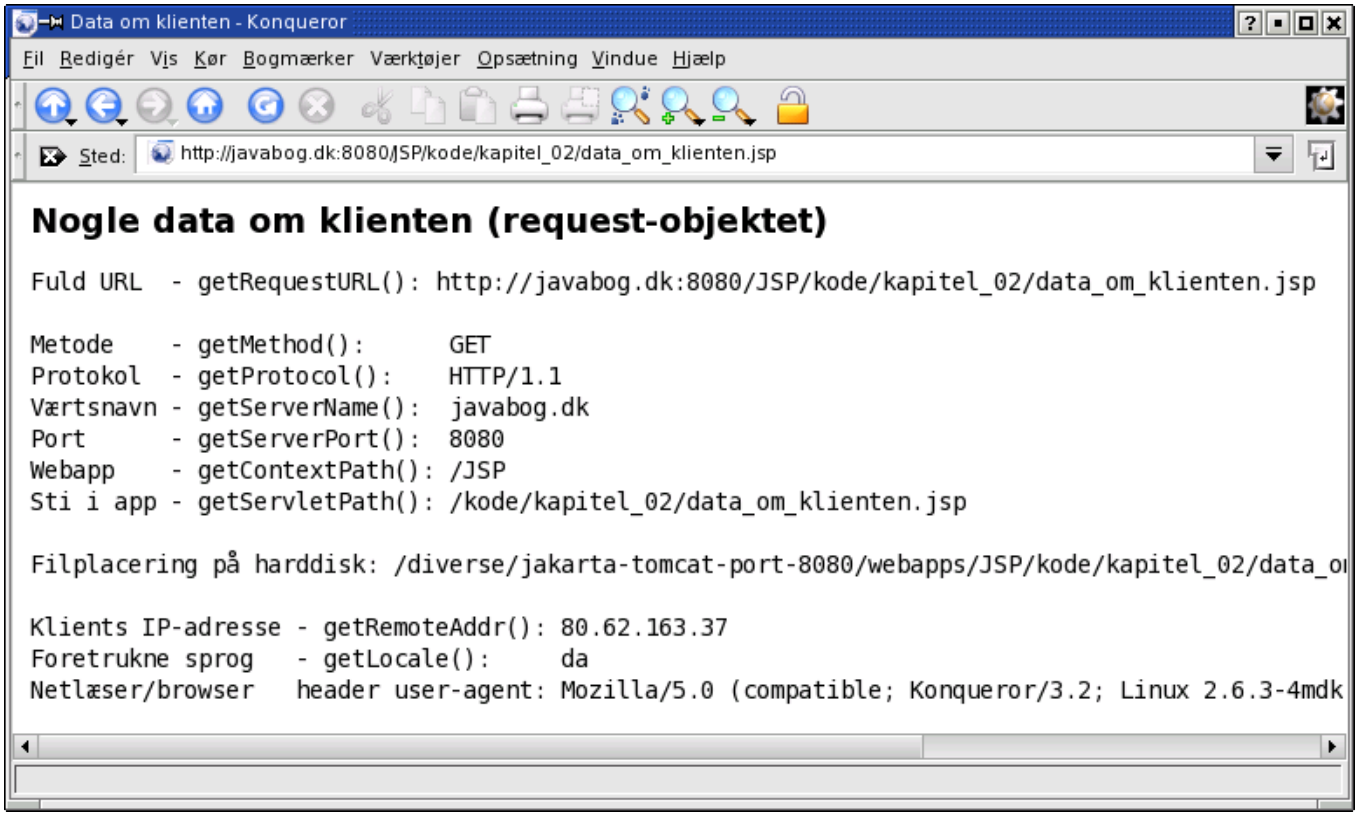
2.4 Data om klienten (request−objektet)

Out−objektet er et eksempel på et objekt, der automatisk findes i alle JSP−sider.

Der findes flere af den slags objekter (i afsnit 4.5, Appendiks: Implicit definerede objekter, er de beskrevet i detaljer), bl.a. et *request*−objekt (beskrevet i afsnit 4.5.1). Dette objekt repræsenterer alt det serveren ved om klienten og med det kan vi få en række ting at vide om klienten og den forespørgsel der blev foretaget.

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

Metode    − getMethod():      <%= request.getMethod() %>
Protokol  − getProtocol():    <%= request.getProtocol() %>
Værtsnavn − getServerName():  <%= request.getServerName() %>
Port      − getServerPort():  <%= request.getServerPort() %>
Webapp    − getContextPath(): <%= request.getContextPath() %>
Sti i app − getServletPath(): <%= request.getServletPath() %>

Filplacering på harddisk: <%= application.getRealPath(request.getServletPath()) %>

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

Prøv eksemplet på http://javabog.dk:8080/JSP/kode/kapitel_02/data_om_klienten.jsp.



2.5 Kommentarer

Med <%−− og −−%> kan man indsætte kodekommentarer i JSP. Disse fungerer lidt anderledes end HTML−kommentarer, som man laver med <!−− og −−>.

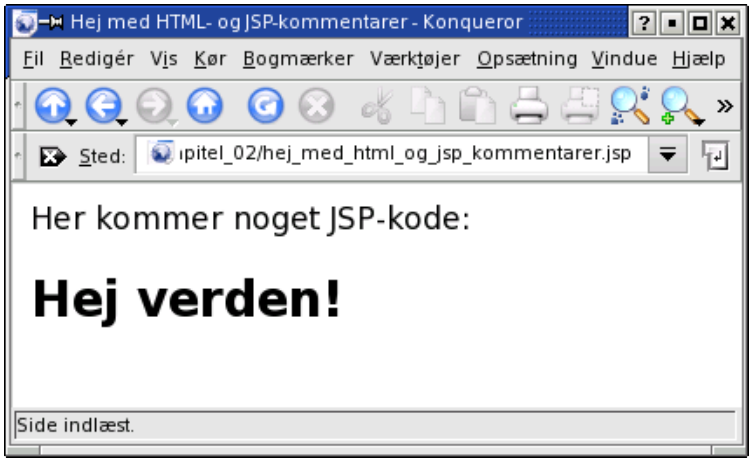
```
<html>
<head><title>Hej med HTML- og JSP-kommentarer</title></head>
<body>
Her kommer noget JSP-kode:

  <% out.println( "<h1>Hej verden!</h1>" ); %>

<!-- HTML-kommentar
  <% out.println( "To plus to er: " ); %>
-->

<%-- JSP-kommentar
  <% out.println( 2 + 2 ); %>
--%>

</body>
</html>
```

I bogen her bliver kommentarer skrevet i *kursiv* for læselighedens skyld.

Grunden til at JSP har sin egen måde at lave kommentarer på, er at JSP−kommentarer med <%−− og −−%> (til forskel fra HTML−kommentarer med <!−− og −−>) slet ikke bliver udført på serveren. De bliver også helt udeladt af koden, der sendes til klientens netlæser. Hvis ovenstående side hentes, vil netlæseren derfor kun få tilsendt (det kan ses med 'vis kilde'):

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

  <h1>Hej verden!</h1>

<!-- HTML-kommentar
  To plus to er:
-->


</body>
</html>
```

De almindelige HTML−kommentarer lavet med <!−− og −−> vil netlæseren udelade i fremvisningen, så resultatet er, at netlæseren ikke viser hverken den ene eller den anden slags kommentarer:



2.6 Test dig selv

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

2.7 Resumé

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

2.8 Avanceret

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

2.8.1 Læse filer fra harddisken

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

2.8.2 Erklæring af metoder og blivende variabler

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

2.8.3 Trådsikkerhed i JSP−sider

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

2.8.4 Producere grafik fra JSP

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

2.8.5 Eksempel: JSP−side der danner et JPG−billede

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

2.8.6 Indlejre og nedskalere billeder fra harddisken

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

2.8.7 Opload af filer til server

1Som vi senere skal se, i afsnit 7.3, Avanceret: JSP−siders interne virkemåde, bliver JSP−sider lavet om til en metode i en klasse. Derfor er det meget vigtigt at {− og }−parenteser er balancerede.

2Egentlig er servletter (se afsnit 7.1) mere velegnede til at producere binære data end JSP−sider. I eksemplet kommer således fejlen 'IllegalStateException: getOutputStream() has already been called for this response' i serverens log, selvom eksemplet virker fint.

3 Interaktive sider

En grundig forståelse af de basale emner i dette kapitel forudsættes i resten af bogen. Kapitlet forudsætter kapitel 2, Grundelementer i JSP.

Vi har nu set hvordan man laver dynamiske hjemmesider, men det bliver først rigtigt interessant, når siderne bliver interaktive, d.v.s. at de kan få input fra brugeren og reagere på det.

3.1 Parametre til sider

Måske har du undret dig over at nogle URLer har et spørgsmålstegn efter dokumentnavnet, f.eks.:
http://minesider.dk/dokument.jsp**?id=1234**

Det, der står efter spørgsmålstegnet, kaldes parametre til siden.
I eksemplet herover er der en parameter med navnet "id" og værdien "1234"

Ligeledes med: http://minesider.dk/dokument.jsp**?navn=Jacob**
Her er parameteren "navn" med værdien "Jacob" sendt sammen med forespørgslen.

Er der flere parametre bliver de adskilt med et &−tegn. Her er f.eks både "id" og "navn" sat:
http://minesider.dk/dokument.jsp**?id=1234&navn=Jacob**

3.1.1 Aflæse en parameter fra URLen

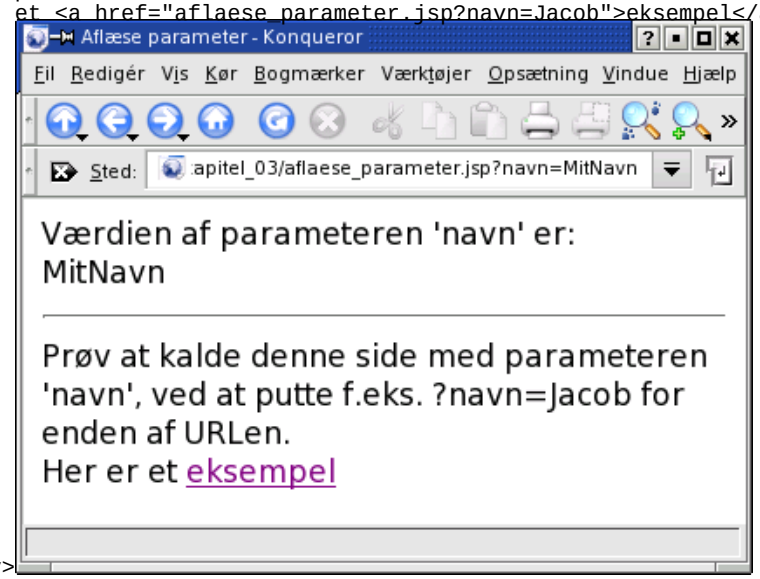
Parametrene i URLen kan aflæses og bruges i JSP−siden. Ønsker man f.eks at aflæse parameteren "navn" skriver man:

```
  String parameterværdi = request.getParameter("navn");
```

Man kalder altså metoden request.getParameter() og angiver navnet på parameteren, man ønsker at aflæse.

Her er et fuldt eksempel på en side, der aflæser parameteren "navn" (det vigtigste i fed):

```
<html>
<head><title>Aflæse parameter</title></head>
<body>
<%
  String parameterværdi = request.getParameter("navn");
  out.print( "Værdien af parameteren 'navn' er: <br>" + parameterværdi );
%>

<hr>

Prøv at kalde denne side med parameteren 'navn',
ved at putte f.eks. ?navn=Jacob for enden af URLen. <br>
Her er et <a href="aflaese_parameter.jsp?navn=Jacob">eksempel</a>
```




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

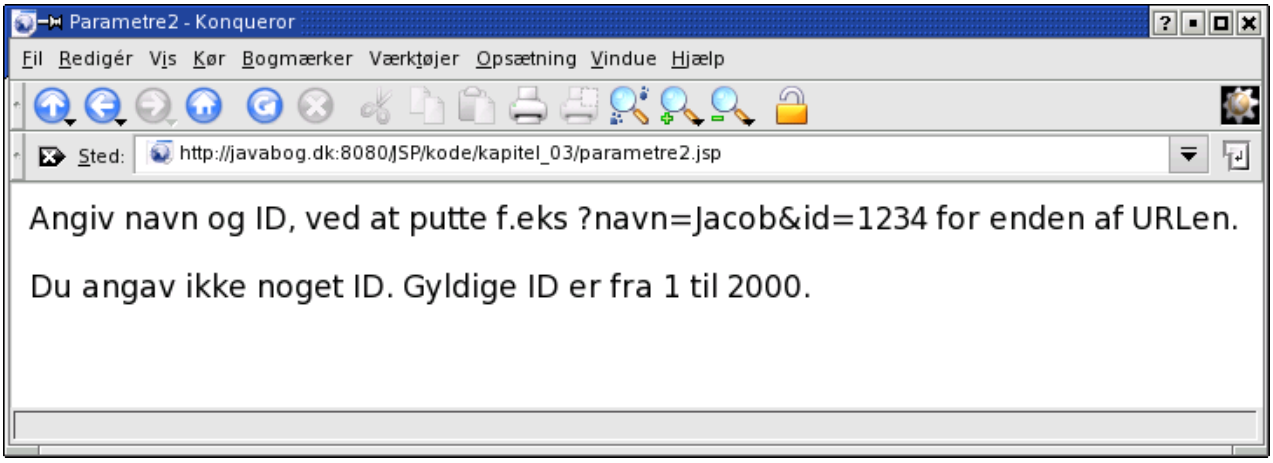
Øvelse

Prøv at køre dette eksempel, hvor du angiver forskellige parametre i URLen. Du kan også lege med eksemplet ved at gå ind på adressen:
http://javabog.dk:8080/JSP/kode/kapitel_03/aflaese_parameter.jsp?navn=MitNavn

3.1.2 Arbejde med parametrene

Det følgende eksempel fanger genkender parametre, nemlig "id" og "navn".

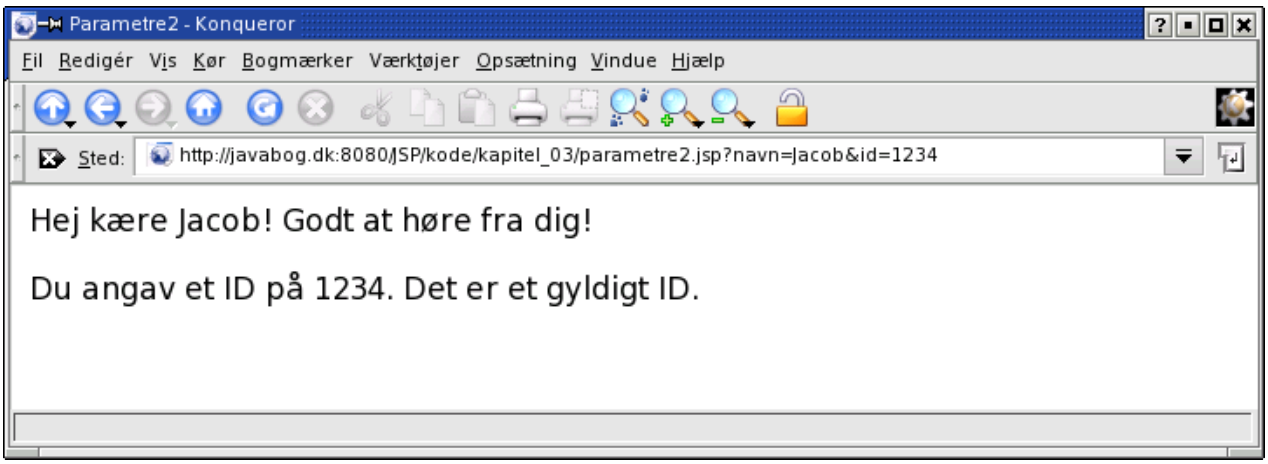
Parametrene gemmes først i nogle interne variabler (af typen String), hvorefter de bruges.



Først undersøges om parameteren "navn" overhovedet er sat til noget (med en if−sætning).

```
<html>
<head><title>Parametre2</title></head>
<body>
<%
  String navnParameter = request.getParameter("navn");
  String idParameter = request.getParameter("id");
  if (navnParameter == null) {
    %>
      Angiv navn og ID, ved at putte f.eks ?navn=Jacob&id=1234
      for enden af URLen.<p>
    <%
  } else {
    %>
      Hej kære <%= navnParameter %>! Godt at høre fra dig!<p>
    <%
  }

  if (idParameter == null) {
    %>
      Du angav ikke noget ID. Gyldige ID er fra 1 til 2000.
    <%
  } else {
    int id = Integer.parseInt(idParameter);
    %>
      Du angav et ID på <%= id %>.
    <%
    if (id <= 0) out.print("Det er for lille.");
    else if (id >2000) out.print("Det er for stort.");
    else out.print("Det er et gyldigt ID.");
  }
%>
</body>
</html>
```

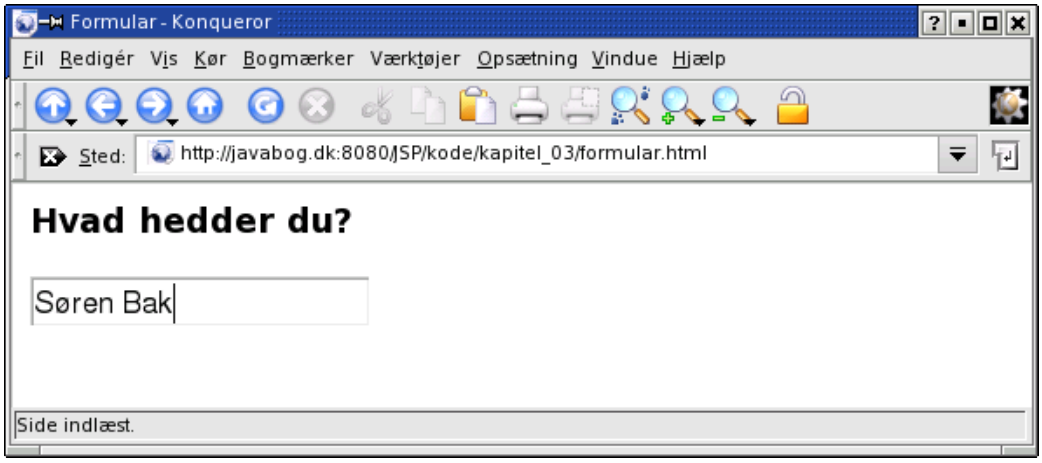
Tilføjer brugeren nogle parametre til URLen svarer serveren med:



3.2 HTML−formularer

Herunder beskrives hvordan man opbygger en formular, som brugeren udfylder, og hvordan brugerens data derefter kan behandles af serveren.

En HTML−formular er en stump HTML−kode, der indeholder et indtastningsfelt, afkrydsningsfelt eller lignende, som brugeren har mulighed for at påvirke. For eksempel:



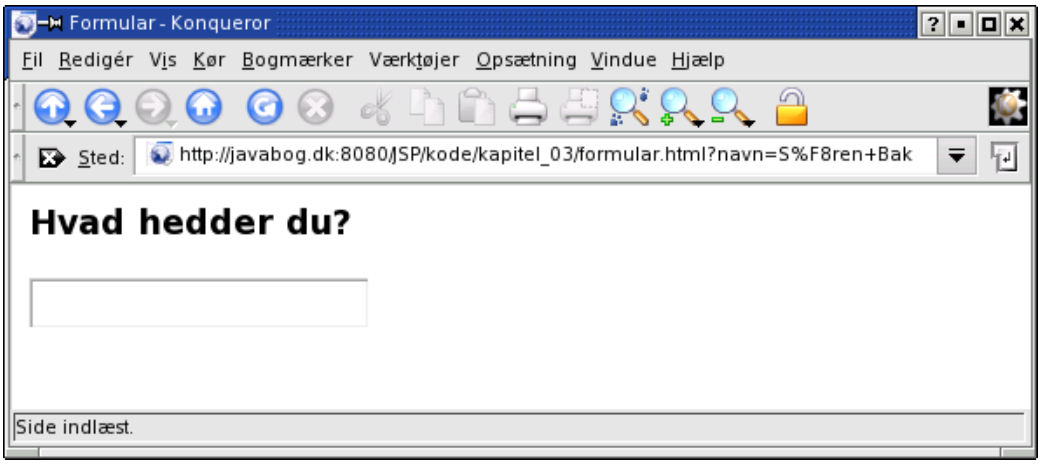
På billedet har brugeren udfyldt indtastningsfeltet med navnet 'Søren Bak'.

HTML−koden til at lave en formular er et <form>−element med en række <input>−elementer af forskellig type (her blot et enkelt af typen "text").

Ovenstående kunne have HTML−koden:

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

Udfylder brugeren formularen og trykker retur vil netlæseren anmode om den samme side igen, men denne gang er indholdet af formularen tilføjet som parametre til URLen:



Læg mærke til, hvordan værdien af parameteren bliver indkodet ('%F8' for 'ø' og et '+' for mellemrum) i URLen af netlæseren. Den bliver automatisk afkodet igen af webserveren når den aflæses fra JSP−siden.

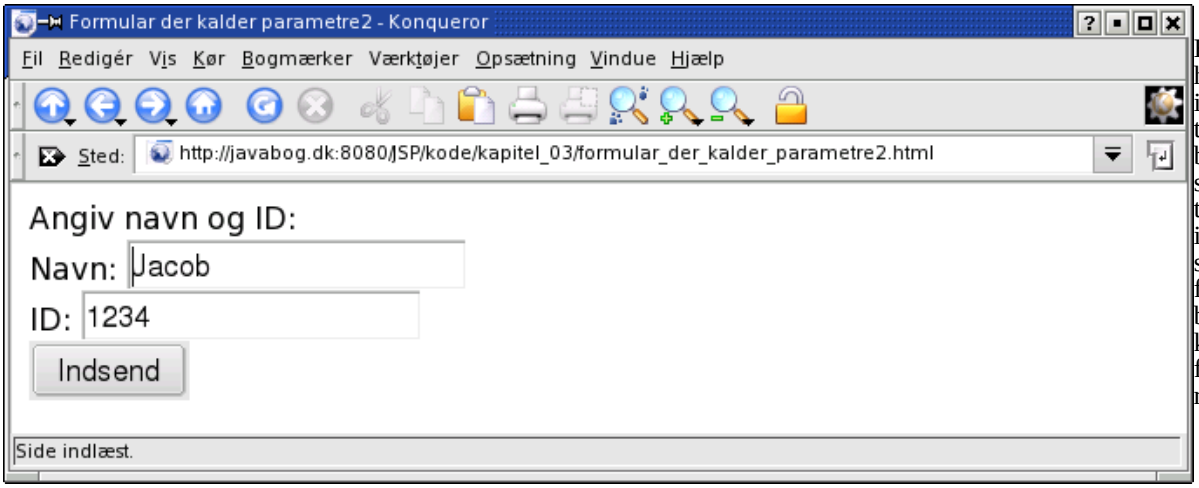
3.2.1 Et lidt større eksempel

Til <form>−koden kan man tilføje en action−attribut, der bestemmer hvilken side der skal anmodes om (sammen med parametrene), når brugeren indsender formularen (ved at trykke retur). Skriver man ikke nogen action−attribut i <form>−koden vil anmodningen blive sendt til den samme side.

Her er et eksempel med en action−attribut:

```
<html>
<head><title>Formular der kalder parametre2</title></head>
<body>
  Angiv navn og ID:<br>
  <form action="parametre2.jsp">
    Navn: <input type="text"   name="navn">               <br>
    ID:   <input type="text"   name="id"   value="1234">  <br>
          <input type="submit" name="OK"   value="Indsend">
  </form>
</body>
</html>
```

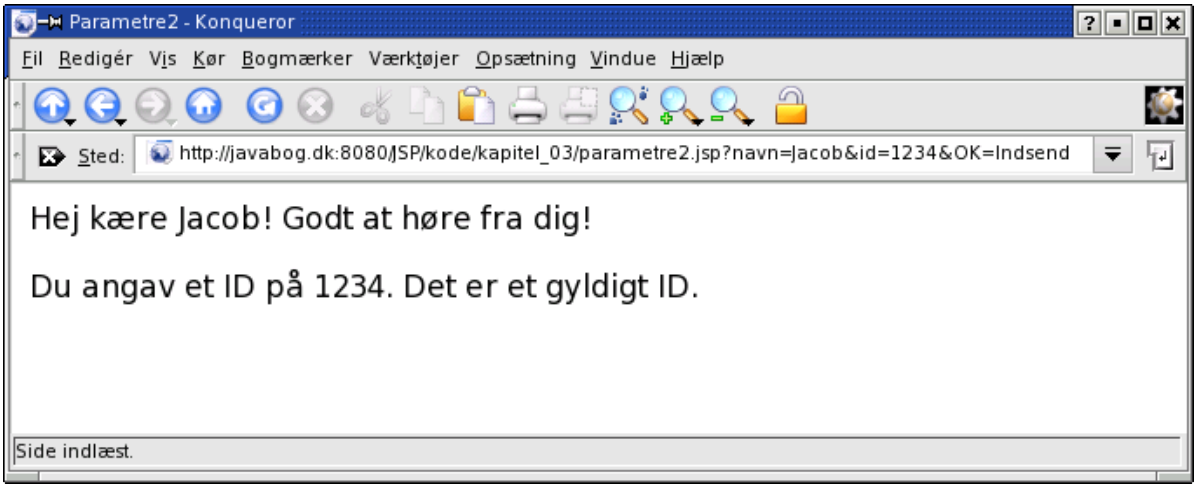
Læg mærke til, at et felt kan være udfyldt på forhånd (value="1234" i indtastningsfeltet med name="id"). Selvom en formular er udfyldt på forhånd kan brugeren godt redigere i feltet og give det en anden værdi.



Bemærk også hvordan input−felter med type="submit" bliver til knapper, som brugeren kan trykke på for at indsende (eng.: submit) formularen (så kan brugeren trykke på knappen i stedet for at trykke retur).

Som tekst der vises på knappen bruges <input>−feltets værdi (her er value="Indsend").

Når brugeren trykker på Indsend−knappen sendes han hen til siden parametre2.jsp (formularen har jo action="parametre2.jsp"):



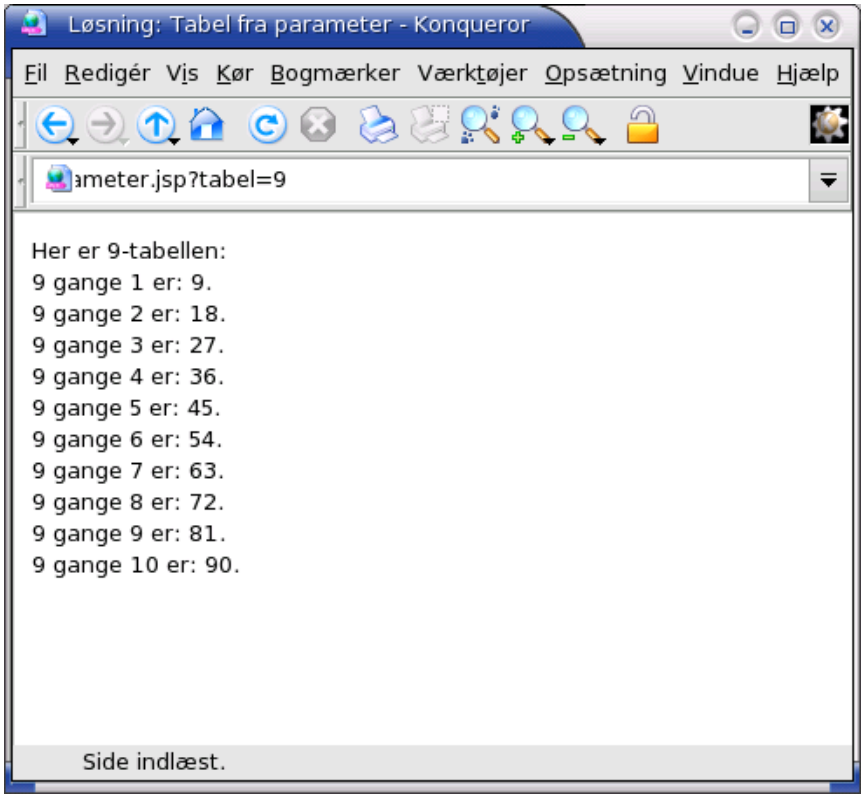
I HTML−koden var der 3 input−felter, med name= hhv. "navn", "id" og "OK". Derfor overføres disse 3 parametre med deres værdier, som navn=Jacob&id=1234&OK=Indsend, der er tilføjet til URLen i adresselinjen.

Den side, der indeholder formularen, behøver egentlig ikke være en JSP−fil. Den behøver ikke engang ligge på den samme webserver som den side, der modtager formulardataene[1].

Også Indsend−knappen (der jo også er et <input>−felt) har fået sin værdi overført som parameter (det kan være rart til at skelne, hvis der er flere indsend−knapper i formularen).

3.2.2 Opgave

1. Ændr i eksemplet Syvtabellen i afsnit 2.2.1 til at aflæse parameteren "tabel", der skal være et nummer, der angiver hvilken tabel der skal udskrives (sådan at man, ved at give parameteren tabel=9, får udskrevet ni−tabellen, som vist til højre).



2. Lav en HTML−formular i JSP−siden, så brugeren kan indtaste hvilken tabel, hun ønsker at få vist.

Løsning

Her er et forslag til løsning:

```
<html>
<head><title>Løsning: Tabel fra parameter</title></head>
<body>
<p>
  Indtast hvilken tabel du ønsker at få vist:<br>
  <form>
    <input type="text" name="tabel" value="9">
  </form>

<%
  String tabelStr = request.getParameter("tabel");
  if (tabelStr != null)
  {
    int tabel = Integer.parseInt(tabelStr);
    out.println("<br>Her er "+tabel+"−tabellen:<br>");

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

3.2.3 Lave HTML og behandle input med samme side

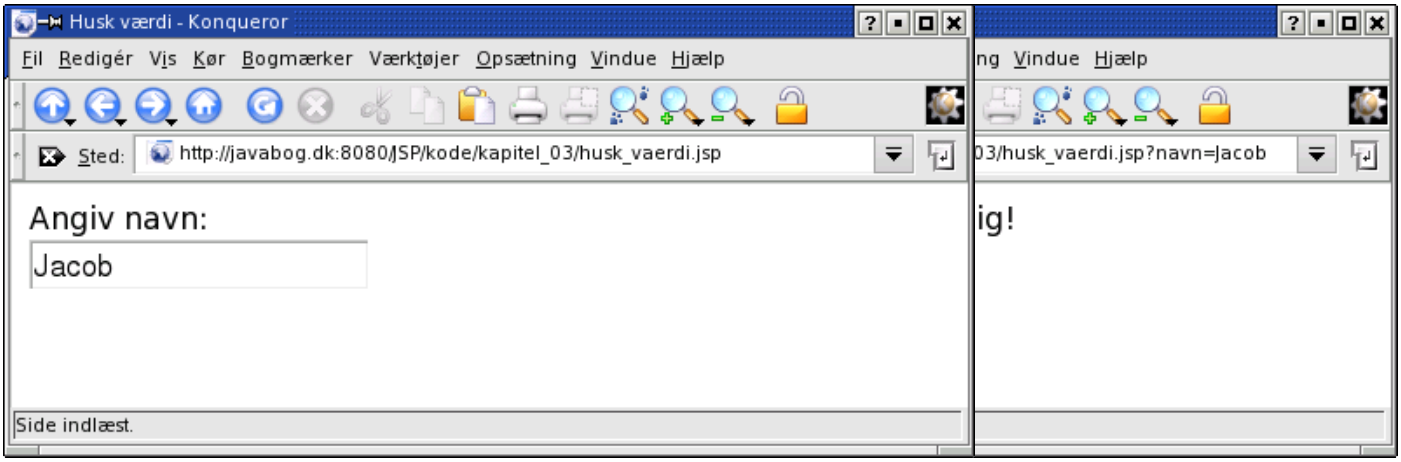
Ofte vil det være den samme JSP−side, der producerer formularen og som behandler brugerens indtastninger i formularen.

Lad os for eksempel lave en side, der husker værdierne i den gamle formular:

husk_vaerdi.jsp

```
<html>
<head><title>Husk værdi</title></head>
<body>
<%
  String navnet = request.getParameter("navn");

  if (navnet == null) {
    %>
      Angiv navn:
      <form>
        <input type="text" name="navn">
      </form>
    <%
```

```
  } else {
    %>
      Hej kære <%= navnet %>! Godt at høre fra dig!<br>
      <form>
        <input type="text" name="navn" value="<%= navnet %>">
      </form>
    <%
  }
%>
</body>
</html>
```



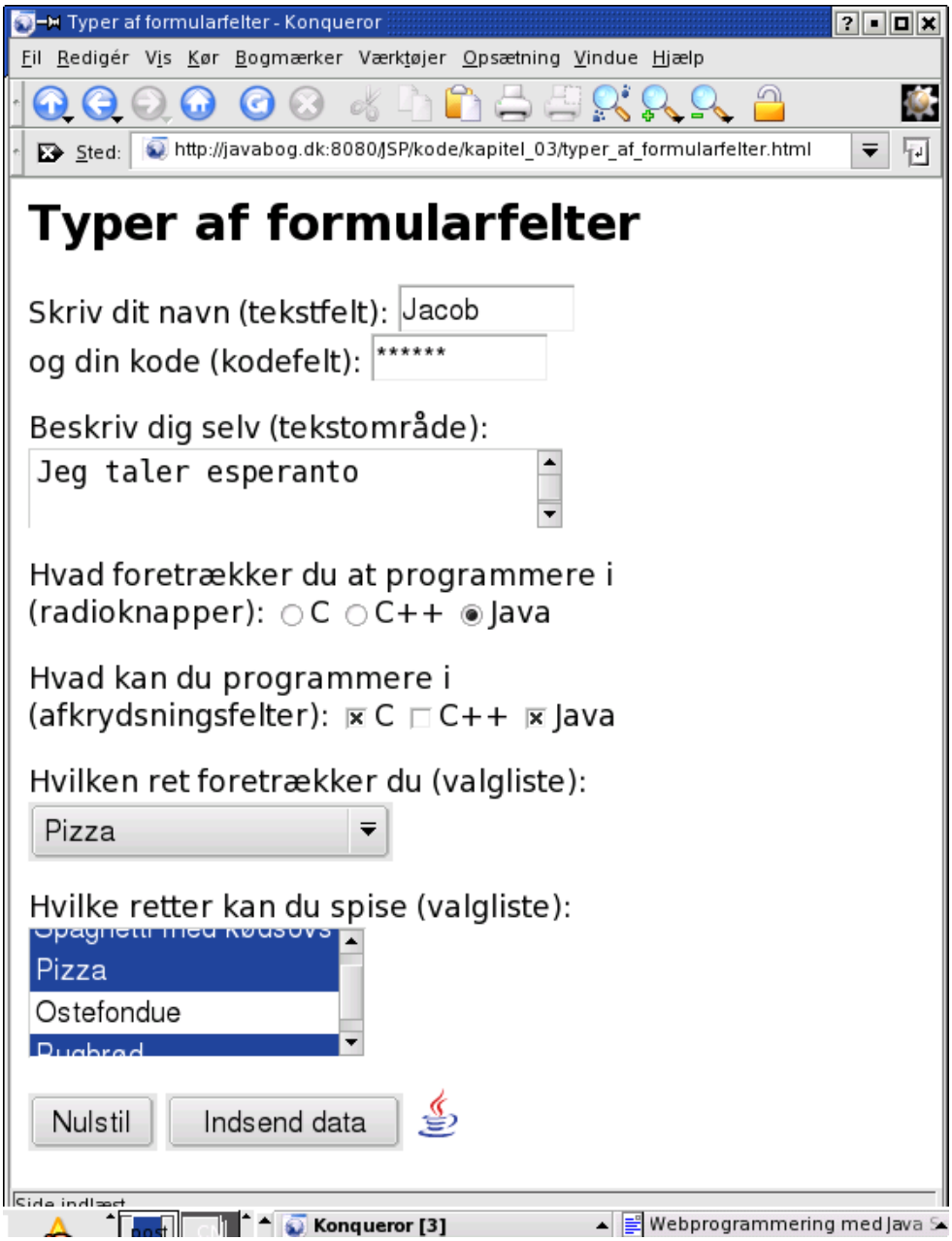
Her er action−attributten til <form> overflødig, da det er den samme side, der får input (den kunne dog sættes med <form action="husk_vaerdi.jsp">).

3.2.4 De almindelige typer af formularfelter

Alle de almindelige slags formularfelter er beskrevet i afsnit 3.3, Appendiks: Typer af formularfelter. Her er en HTML−side med dem. Sammenlign med billedet på næste side, der viser hvordan det ser ud i en netlæser (Konqueror under Linux).

```
<html>
<head><title>Typer af formularfelter</title></head>
<body>
<h1>Typer af formularfelter</h1>
<form action="parametre3.jsp">

<p>Skriv dit navn (tekstfelt):
  <input type="text" name="navn" value="Jacob" size="10">
   <br>og din kode (kodefelt):
  <input type="password" name="kode" value="abcdef" 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 esperanto
  </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="checked">Java
</p>

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

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

<p>Hvilke retter kan du spise (valgliste): <br>
  <select name="kan_spise" size="3" multiple="multiple">
```

```
    <option value="spa" selected="selected">Spaghetti med kødsovs</option>
    <option value="piz" selected="selected">Pizza</option>
    <option value="ost">Ostefondue</option>
    <option value="rug" selected="selected">Rugbrød</option>
  </select>
</p>

<p>
  <input type="reset" value="Nulstil">
  <input type="submit" name="indsendKnap" value="Indsend data">
  <input type="image" src="sun_java.png" name="javaKnap" value="kaffekop">
</p>

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



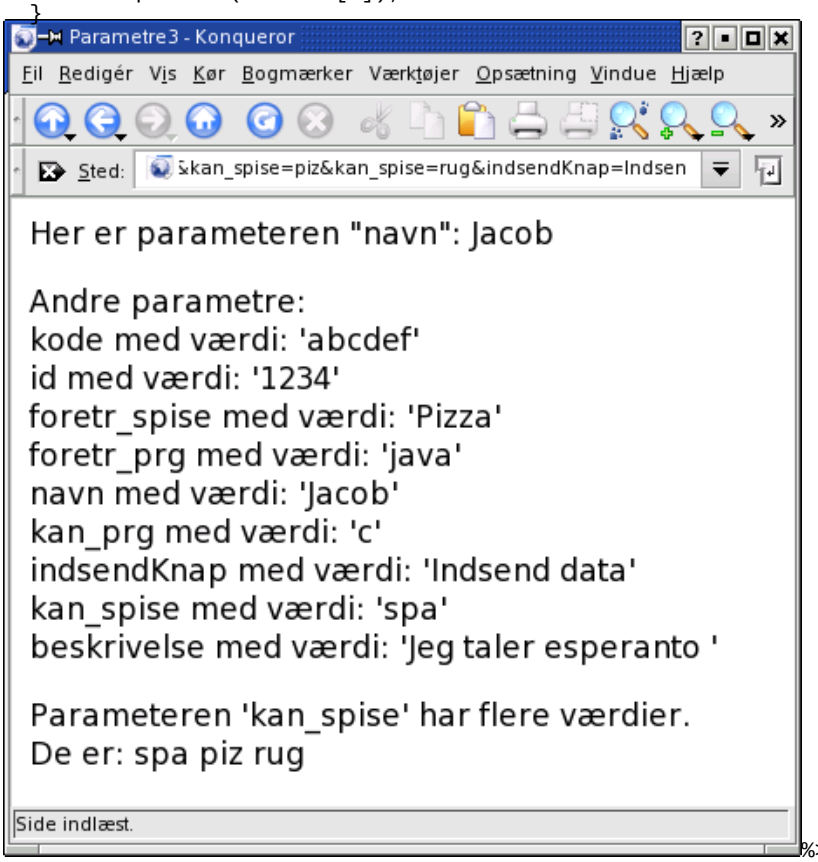
3.2.5 Aflæse alle parametre i en formular

Det følgende aflæser alle parametrene i ovenstående formular og udskriver dem.

parametre3.jsp (JSTL−eksempel i afsnit 6.1.4)

```
<html>
<head><title>Parametre3</title></head>
<body>

<p>
Her er parameteren "navn": <%= request.getParameter("navn") %><br>
</p>

<p>
```

```
Andre parametre:<br>
<%
  // udskriv alle parametrene
  java.util.Enumeration enumeration = request.getParameterNames();
  while (enumeration.hasMoreElements()) {
    String parameternavn = (String)enumeration.nextElement();
    String parameterværdi = request.getParameter(parameternavn);
    out.println(parameternavn+" med værdi: '"+parameterværdi+"'<br>");
  }
%>
</p>

<p>
Parameteren 'kan_spise' har flere værdier.<br>
De er:
<%
  String[] værdier = request.getParameterValues("kan_spise");
  if (værdier != null) {
    for (int i = 0; i < værdier.length; i++)
      out.println(værdier[i]);
  }
```




```
%>
</p>

</body>
</html>
```

3.2.6 Videre læsning

For en grundigere gennemgang af, hvilke forskellige felter, der kan findes i en formular, henvises til f.eks. artiklen "Lær at lave formularer" på http://html.dk/artikler/00011.

3.2.7 Opgaver

1. Lav en valutaberegnings−webside, der kan omregne fra US dollar til euro.
2. Udvid valutaberegnings−websiden med en valgliste, så brugeren kan vælge mellem flere valutaer
3. Lav en tekstanalyse−webside med et indtastningsfelt (textarea), hvor brugeren kan indtaste (eller indsætte fra et andet dokument) en tekst, som websiden derefter behandler.
   Websiden kan f.eks. oplyse antallet af tegn i teksten og hvordan strengen ser ud skrevet bagfra (vink.: Brug new StringBuffer(teksten).reverse().toString() for at vende strengen om).
4. Udvid tekstanalyse−websiden, så brugeren kan taste en ordlængde ind, hvorefter siden tæller antallet af ord med netop den længde.

3.3 Appendiks: Typer af formularfelter

Her er en oversigt over de mest almindelige typer af formularfelter:

HTML−kode

Betydning

`<input type="text">`

Indtastningsfelt (én linje)

`<input type="password">`

Kodefelt − indtastningsfelt, hvor der vises stjerner i stedet for den indtastede tekst

`<input type="hidden">`

Skjult felt. Vises ikke for brugeren, men har alligevel en værdi, se afsnit 3.6.2, Skjulte felter i formularer.

`<textarea>`

Tekstområde på flere linjer

`<input type="checkbox">`

Afkrydsningsfelt

`<input type="radio">`

Radioknapper, d.v.s. afkrydsningsfelter, der gensidigt udelukker hinanden (de skal have samme name−attribut)

`<select>`

Valgliste. Hver mulighed markeres med en <option>−indgang. Med size−attributten kan man lade flere elementer i listen blive vist på en gang. Svarer funktionelt til et sæt radioknapper). Med multiple−attributten kan flere indgange vælges (så den funktionelt svarer til et sæt afkrydsningsfelter).

`<input type="image">`

Billede − trykker man på billedet indsendes formularen med x− og y−koordinater på, hvor på billedet brugeren trykkede

`<input type="submit">`

Indsendingsknap − trykker man på knappen indsendes formularen med knappens navn og værdi som parameter.

`<input type="reset">`

Nulstiller formularen til dens udgangspunkt, d.v.s. annullerer alle brugerens ændringer.

3.4 Test dig selv

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

3.5 Resumé

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

3.6 Avanceret

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

3.6.1 Indkode data i en URL (URL−indkodning)

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

3.6.2 Skjulte felter i formularer

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

3.6.3 Bruge skjulte felter til at etablere et forløb

3.6.4 Skjule parametrene (POST−metoden)

3.6.5 Cookier

3.6.6 Sætte cookier

3.6.7 Aflæse cookier

3.7 Avanceret: HTTP−protokollen

3.7.1 Eksempel på kommunikation

3.7.2 Formulardata med GET−metoden

3.7.3 Formulardata med POST−metoden

3.7.4 Cookier

3.7.5 Øvelse

3.7.6 Sende mere data til klienten løbende

3.7.7 Eksempel: Syvtabellen langsomt

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

3.7.8 Eksempel: Følge med i serverens logfil

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

1Da angiver man blot en absolut URL, f.eks.:
<form action="http://minesider.dk/parametre2.jsp">

2Og derved kan man så ikke mere foretage omdirigeringer, j.v.f. afsnit 3.7 og afsnit 4.3.

4 Videre med JSP

En overfladisk forståelse af emnerne i dette kapitel forudsættes i det meste af bogen. Kapitlet forudsætter kapitel 3, Interaktive sider.

4.1 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 nyttig til at huske data, der skal følge brugeren.

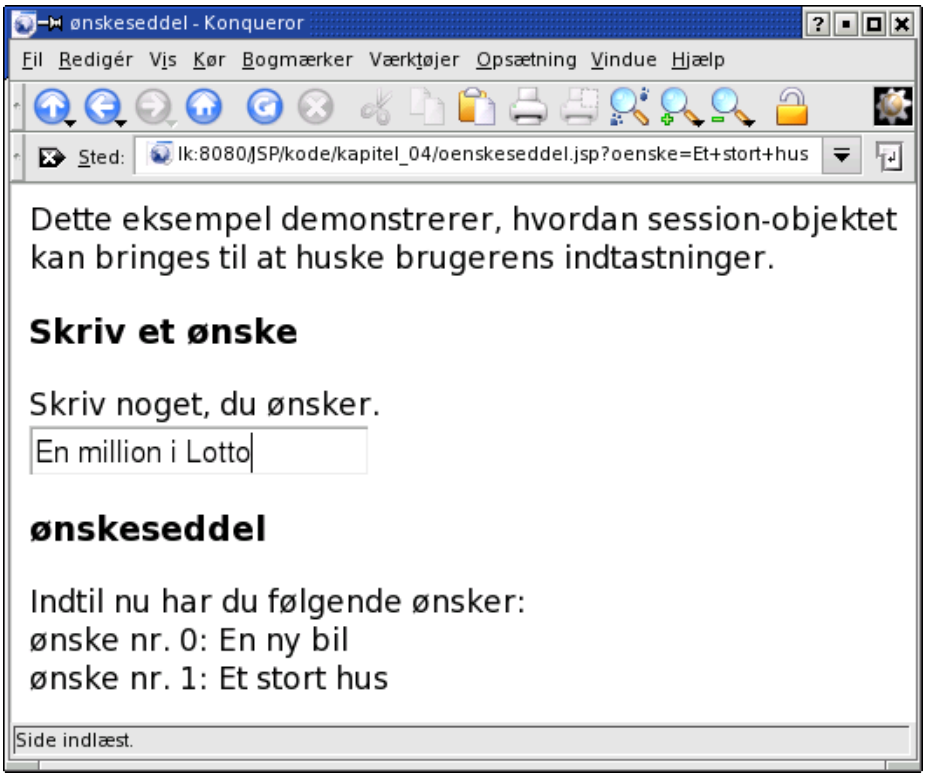
Med den kan man gemme og hente oplysninger i løbet af en brugers besøg. Det kunne f.eks. være om brugeren har logget ind, et bruger−ID eller nogle oplysninger om hvilke valg, brugeren har foretaget. I en e−handels−applikation ville sessionsobjektet også være det helt rigtige at bruge til at huske varerne i brugerens indkøbskurv.

De vigtigste metoder i session−objektet er beskrevet i afsnit 4.5.4.

4.1.1 Eksempel: En ønskeseddel

Det følgende eksempel lader brugeren indtaste nogle ønsker. Ønskerne huskes i en liste (af type ArrayList), der gemmes i sessionsobjektet (under navnet 'ønsker').

```
<%@ page language="java" import="java.util.*" %>
<html>
<head><title>Ønskeseddel</title></head>
<body>
Dette eksempel demonstrerer, hvordan session-objektet kan bringes til at
huske brugerens indtastninger.

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

  if (ønsker == null) {                    // hvis listen ikke findes:
    ønsker = new ArrayList();              // opret den
    session.setAttribute("ønsker", ønsker);  // og registrer den under "ønsker"
  }

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

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

  }
%>
</body>
</html>
```



4.1.2 Sessioner er individuelle

Forestil dig, at 20 personer samtidigt klikker rundt på det samme websted − f.eks. en e−butik. Oplysningerne om, hvad deres indkøbskurve indeholder, kan med fordel gemmes i sessionsobjektet.

For at kunne identificere brugerne i forhold til hinanden, sådan at to brugere ikke ved et uheld får byttet session (og dermed varer i indkøbskurven!) bruger serveren et unikt ID, som gemmes i en HTTP cookie (en lille tekstfil på brugerens PC, se afsnit 3.6.5). Brug af sessioner kræver derfor normalt, at brugeren har slået understøttelse af cookier til i sin netlæser (ellers må man ty til 'URL Rewriting' beskrevet i afsnit 4.1.4).

4.1.3 At kassere en session

Vil du smide sessionen helt væk, kan det gøres med kaldet

```
session.invalidate()
```

Herefter er sessionen og alle dens data væk.

Hvis den pågældende bruger besøger en side igen, vil hun blive tildelt en ny, tom session.

4.1.4 Avanceret: URL Rewriting

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

4.2 Eksempel: Login med adgangskode

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

4.2.1 Inkludering af kodefragmenter

I det foregående eksempel skulle alle beskyttede sider have et ovenstående tjek for, om brugeren er logget korrekt ind.

Det kunne derfor nok betale sig at have denne stump kode i en separat fil:

logintjek.jsp

```
<%
  // Filnavn: logintjek.jsp

  // 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");
  }
%>
```

Herefter kan siderne blot inkludere fragmentet logintjek.jsp for at få tjekket om brugeren er logget ind:

login4.jsp

```
<%@ include file="logintjek.jsp" %>
<html>
<head><title>login4</title></head>
<body>

<h1>En anden beskyttet side</h1>
Denne tekst kan du kun se, hvis du er logget korrekt på.

</body>
</html>
```

4.3 Omdirigering

Ovenstående eksempel viser også, hvordan man omdirigerer brugeren til en anden side:

```
  response.sendRedirect("login1.html");
```

Brugeren vil da få siden login1.html i stedet for den side han egentlig spurgte om.

Omdirigering forudsætter, at der endnu ikke er sendt nogen data til brugeren. For JSP−sider sendes data normalt først, når afviklingen af siden er afsluttet (output er buffered), så en beslutning om omdirigering bør ske tidligt[1] på siden.

Omdirigering kan faktisk gøres på to måder for JSP−sider: med response.sendRedirect():

```
<% response.sendRedirect("login1.html") %>
```

eller med JSP−koden <jsp:forward />:

```
<jsp:forward page="login1.html"/>
```

Forskellen mellem de to måder er måden, som omdirigeringen til den nye side foregår på: I det første tilfælde er klienten involveret i omdirigeringen, i det andet tilfælde sker omdirigeringen internt på serveren, uden at klienten ved det.

Modsat hvad man måske umiddelbart skulle tro, bliver al javakoden *efter* (d.v.s. neden under) en omdirigering faktisk udført, selvom outputtet fra siden skal kasseres[2].

4.3.1 Klient−omdirigering (response.sendRedirect())

Med response.sendRedirect() sendes et speciel svar til netlæseren om, at den side, den lige har spurgt på, midlertidigt er flyttet et andet sted hen, hvorefter netlæseren vil spørge en gang til på den nye adresse.

Teknisk set sker der det, at serveren udnytter HTTP−protokollen (beskrevet i afsnit 3.7) til at omdirigere klienten. Når klienten spørger på login3.jsp med:

```
GET /JSP/kode/kapitel_04/login3.jsp HTTP/1.1
Host: javabog.dk:8080
```

vil serveren, i stedet for at kvittere med et '200 OK' og sende data som normalt, svare med et '302 Moved Temporarily' og fortælle klienten at den 'nye adresse' er login1.html:

```
HTTP/1.1 302 Moved Temporarily
Location: http://javabog.dk:8080/JSP/kode/kapitel_04/login1.html
```

Netlæseren reagerer på '302 Moved Temporarily' ved omgående at sende en ny anmodning, der spørger på den nye adresse:

```
GET /JSP/kode/kapitel_04/login1.html HTTP/1.1
Host: javabog.dk:8080
```

Herefter forløber HTTP−kommunikationen som den plejer.

Netlæseren har været involveret i omdirigeringen og viser den nye URL i adresselinjen. Parametrene til den oprindelige anmodning sendes *ikke* igen sammen med den nye anmodning, de mistes.

Omdirigering med JavaScript

JavaScript kan også bruges til at omdirigering. Følgende omdirigerer også til login1.html:

```
<html>
<head><title>Omdirigering med JavaScript</title></head>
<body>
<script>document.location="/JSP/kode/kapitel_04/login1.html"</script>
</body>
</html>
```

JavaScript udføres i brugerens netlæseren og kan ikke bruges til sikkerhedsforanstaltninger, j.v.f. afsnit 8.5, Sikkerhed i en webapplikation (det kan f.eks. deaktiveres af brugeren).

4.3.2 Server−omdirigering (<jsp:forward />)

Med <jsp:forward /> omdirigeres anmodningen *internt* i serveren. Det vil sige at den oprindelige anmodning klienten foretog (med de samme parametre etc.), besvares af en anden side i serveren.

Svaret sendes derefter tilbage til klientens netlæser, der ikke ved, at der skete en omdirigering. Den viser derfor den oprindelige URL i adresselinjen, ikke den nye adresse.

F.eks. kunne det være at klienten skulle omdirigeres til login−siden:

```
<jsp:forward page="login2.jsp"/>
```

Denne omdirigeringsmetode tillader at sende parametre videre til modtagersiden, da det samme request−objekt genbruges.

Man kan endda også *tilføje* parametre. Hvis brugeren eksempelvis kom ind på en side hvor der i JSP−siden, serveren udførte, stod:

```
<jsp:forward page="login2.jsp">
  <jsp:param name="brugernavn" value="Jacob"/>
  <jsp:param name="adgangskode" value="hemli"/>
</jsp:forward>
```

ville han blive automatisk logget ind (i øvrigt nok ikke videre hensigtsmæssigt).

Server−omdirigering fra javakode

Fra en servlet, eller fra Java−koden i en JSP−side kunne man skrive:

```
request.getRequestDispatcher("login2.jsp").forward(request,response);
```

4.4 Appendiks: Almindelige JSP−koder

Her er en oversigt over almindelige JSP−koder, som kan være nyttig til senere opslag.

JSP−kode	Betydning
`<% kode %>`	Javakode der udføres på serveren hver gang siden hentes.
`<%= udtryk %>`	Et udtryk. Værdien af udtrykket vil blive beregnet og indsat i stedet for koden. Eksempel (se også afsnit 2.2.1 Indlejrede java−udtryk): `Syv gange 2 er <%= 7*2 %>`
`<%−− kommentar −−%>`	En kommentar. Vil ikke blive udført af serveren og vil aldrig blive sendt til klienten. Se også afsnit 2.5 Kommentarer.
`<%! erklæring %>`	Variabel− og metodeerklæringer. Disse oprettes/defineres én gang når siden indlæses (se afsnit 2.8.2). Bruges sjældent.
`<%@ page ... %>`	Side−direktivet, der indeholder information om siden, f.eks.: language="java" (beskrivelse af sproget i siderne) import="java.util.*" (import af klassedefinitioner) session="false" (webserveren skal ikke holde styr på brugersessioner) contentType="text/html;charset=UTF−8" (indholdstype og evt. tegnsæt) errorPage="fejl.jsp" (opstår en undtagelse så vis fejl.jsp i stedet) isErrorPage="true" (hvis denne side er en fejlmeddelelsesside − får defineret objektet *exception*, der beskriver fejlen. Se også afsnit 10.4.5).
`<%@ taglib ... %>`	Et JSP−kodebibliotek (eng.: Tag library). En kraftfuld måde at definere og anvende foruddefinerere JSP−funktioner på (se afsnit 6).
`<%@ 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"/>`
`<jsp:forward ... />`	Omdirigerer til en anden side (som bliver kaldt i stedet, se afsnit 4.3.2): `<jsp:forward page="hej.jsp"/>`

`<jsp:plugin ... />`	Genererer HTML−kode til en plugin. Bruges sjældent.
	F.eks. Java−plugin til appletter der skal køre under JDK1.2:
	
	`<jsp:plugin type="applet" code="MinApplet.class"`
	`jreversion="1.2" width="160" height="150" />`

<jsp:...>−koderne anvender XML−syntaks: Koder der begynder med <jsp: skal afsluttes med et /> eller også skal der være en tilsvarende slutkode. Det betyder at f.eks.:

```
<jsp:forward page="login1.html">
```

ikke må stå alene, men **skal** afsluttes af den tilsvarende slut−kode:

```
</jsp:forward>
```

For at slippe for at skrive en masse slut−koder i de (mange!) tilfælde, hvor koden skal afsluttes lige efter at den er startet, er der indført en forkortet skrivemåde:

```
<jsp:forward page="login1.html"/>
```

hvor man afslutter koden med /> til sidst. Dette svarer altså til

```
<jsp:forward page="login1.html"></jsp:forward>
```

4.5 Appendiks: Implicit definerede objekter

Der findes en række implicit definerede objekter, som man altid har adgang til i en JSP−side. I det følgende vil disse objekter og deres vigtigste metoder blive beskrevet.

4.5.1 request − anmodningen fra klienten

Objektet request (af type HttpServletRequest i pakken javax.servlet.http) repræsenterer anmodningen fra klienten (i en servlet bliver request−objektet overført i doGet()−metoden).

Objektet kan bruges til at få information om klienten (se eksempel i afsnit 2.4), formulardata fra klienten (se afsnit 3.2.5) og informationer om gemte cookier (se afsnit 3.6.5).

String **getRequestURL**()
Giver den fulde URL på siden, der anmodes om, med maskinnavn og med evt parametre

String **getMethod**()
Giver anmodningsmetoden, som normalt er "GET". For formularer kan "POST" også forekomme (en tredje mulighed, "PUT" bruges til at sende filer til serveren). Se afsnit 3.6.4.

String **getProtocol**()
Giver versionen af HTTP−protokollen klienten har anvendt, typisk "HTTP/1.1"

String **getServerName()**
Giver værtsnavnet (eng.: host name), f.eks: "javabog.dk".

String **getContextPath()**
Giver stien til webapplikationens rod.

String **getServletPath()**
Giver stien på serveren på siden, der anmodes om (evt. parametre er fjernet), relativt til webapplikationens rod. Med application.getRealPath(request.getServletPath()) kan man finde den absolutte sti og filnavnet til, hvor en JSP−side ligger fysisk på harddisken (vist i afsnit 2.8.1).

String **getRemoteAddr**() og **getRemoteHost**()
Giver adressen (som IP−nummer eller evt med navn) på klienten, der foretager anmodningen

String **getLocale**()
Giver brugerens foretrukne sprog (se eksempel på brug i afsnit 2.4 og afsnit 13.2)

String **getParameter**(String parameternavn)
Giver værdien af en parameter fra klienten − typisk fra en udfyldt formular, eller fra URLen. F.eks hvis en side bliver kaldt som side.jsp?x=a siges det, at den har parameteren x med værdien a og getParameter("x") vil returnere strengen "a".

String[] **getParameterValues**(String parameternavn)
Giver et array af værdier for et parameternavn. HTTP−protokollen tillader, at parametre har flere værdier. F.eks. vil side.jsp?x=aaa&x=bbb&x=ccc, hvor parameteren "x" har tre værdier, få getParameterValues("x") til at returnere et array med værdier {"aaa", "bbb", "ccc"}. Se afsnit 3.2.5.

Enumeration **getParameterNames**()
Giver en opremsning (eng.: enumeration) hvor hvert element er navnet på en parameter.
F.eks vil getParameterNames() give en opremsning med "x", "y" og "y2" for en side kaldt med: side.jsp?x=aaa&y=bbb?y2=ccc.
Se et eksempel på brug i afsnit 3.2.5.

Cookie[] **getCookies**()
Giver et array af cookier, der er sat hos klienten (netlæseren). Se afsnit 3.6.5, hvor der også er et eksempel.

4.5.2 response − svaret til klienten

Objektet response (af type HttpServletResponse i pakken javax.servlet.http) repræsenterer svaret på anmodningen (i en servlet bliver response−objektet overført i doGet()−metoden). Objektet har information og data til klienten, herunder hvilke cookier der evt. skal sættes i klienten.

De vigtigste metoder er:

void **setContentType**(String indholdstype)
Sætter indholdstypen (MIME−typen). Indholdstypen er fra systemets side sat til "text/html".
Ønsker man at sende andre slags data, f.eks. binære, skal indholdstypen sættes til "image/jpeg" (jpeg−billeder), "audio/x−wav" (wav−lyde) eller "application/x−dit−eget−valg" hvis det er en helt selvopfunden indholdstype. Se et eksempel i afsnit 2.8.4, Producere grafik fra JSP.
Tegnsættet kan også sættes, f.eks. til Unicode (indholdstype="text/html;charset=UTF−8").

void **addCookie**(Cookie cookie)
Sætter eller tilføjer en cookie hos klienten (netlæseren). Se afsnit 3.6.5.

void **sendRedirect**(String url)
Omdirigerer klienten til en anden side, den skal spørge på i stedet for denne side. Adressen i url kan godt være relativ. Relateret til <jsp:forward ... />, se afsnit 4.3, Omdirigering.

4.5.3 out − skrive tekst til klienten

Objektet out (af type JspWriter i pakken javax.servlet.jsp) repræsenterer datastrømmen, der bliver sendt til klienten.

Det bruges lige som System.out:

```
out.println( "<h1>Hej verden!</h1>" );
```

4.5.4 session − objekt der følger den enkelte bruger

Hver bruger får[3] tildelt et session−objekt (af type HttpSession i pakken javax.servlet.http), der repræsenterer brugerens session. Det følger brugeren, lige meget hvilken side han/hun er inde på, og er derfor nyttigt til at huske data, der skal følge brugeren, f.eks om vedkommende er logget ind, sessions−ID og brugeroplysninger.

void **setAttribute**(String nøgle, Object værdi)
Gemmer et objekt under navnet *nøgle*.

Object **getAttribute**(String nøgle)
Henter objektet under navnet *nøgle*.

void **removeAttribute**(String nøgle)
Sletter referencen til objektet under navnet *nøgle*.

String[] **getAttributeNames**()
Giver et array af alle nøgler der er defineret.

long **getLastAccessedTime**()
Giver antal millisekunder siden sessionen sidst var i brug

int **getMaxInactiveInterval**()
Hvor lang tid (i sekunder) der går før sessionen bliver smidt væk, hvis den ikke bruges.

void **setMaxInactiveInterval**(int sekunder)
Sætter hvor lang tid (i sekunder) der går før en session, der ikke bruges, skal smides væk.

void **invalidate**()
Smider sessionen væk. Hvis den pågældende bruger besøger en side igen, vil hun blive tildelt en ny session.

4.5.5 application − fælles for hele webapplikationen

Objektet application (af type ServletContext i pakken javax.servlet) er fælles for alle sider og alle brugere af webapplikationen. Det kan bruges til at huske 'globale' data[4].

Derudover kan den bruges til logning i webserverens log og til at fremfinde initialiseringsparametre fra web.xml, den globale konfigurationsfil for webapplikationen.

void **setAttribute**(String nøgle, Object værdi)
Gemmer et objekt under navnet *nøgle*.

Object **getAttribute**(String nøgle)
Henter objektet under navnet *nøgle*.

void **removeAttribute**(String nøgle)
Sletter referencen til objektet under navnet *nøgle*.

Enumeration **getAttributeNames**()
Giver en opremsning af alle nøgler der er defineret.

void **log**(String besked)

Logger en besked.

void **log**(String besked, Throwable undtagelse)

Logger en fejlmeddelelse og en supplerende undtagelse, hvis stakspor skrives til loggen.

String **getRealPath**(String url_i_webapp)

Giver den rigtige sti (på harddisken) ud fra en URL i webapplikationen. Er nyttigt til at indlæse og behandle filer fra harddisken, der befinder sig i samme webapplikation, se afsnit 2.8.6.
Med application.getRealPath(request.getServletPath()) kan man finde den absolutte sti og filnavnet til hvor en JSP−side ligger fysisk på harddisken (dette er vist i afsnit 2.8.1).

String **getInitParameter**(String navn)

Giver værdien af initialiseringsparameteren *navn* defineret i webapplikationens web.xml.

Enumeration **getInitParameterNames**()

Giver en opremsning af navnene på alle initialiseringsparametrene defineret i web.xml.

Initialiseringsparametre

application−objektet giver også adgang til initialiseringsparametre fra web.xml.

Initialiseringsparametre er meget hensigtsmæssige til f.eks. at lægge navne på databasedrivere og database−URLer og lignende faste oplysninger. Det gøres ved at lave en indgang af typen context−param, f.eks. med databasedriveren, i web.xml:

```
<web−app>
  ...
  <context−param>
    <param−name>dbDriver</param−name>
    <param−value>com.mysql.jdbc.Driver</param−value>
  </context−param>
  ...
</web−app>
```

Nu vil man i en JSP−side kunne hente værdien af initialiseringsparameteren dbDriver med:

```
    String drv = application.getInitParameter("dbDriver");
```

hvorefter strengen drv vil have indholdet 'com.mysql.jdbc.Driver'. Der er et eksempel på dette i afsnit 8.5.8.

Fra en servlet (se afsnit 7.1, Servletter) er application−objektet tilgængeligt med:

```
    ServletContext application = getServletContext();
```

hvorefter værdien fås som i en JSP−side med:

```
    String drv = application.getInitParameter("dbDriver");
```

I afsnit 9.5, Eksempel: Login og brugeroprettelse, kan man se hvordan en ekstern klasse (en javabønne, vist i afsnit 9.5.2) kan bruge application−objektet til af få initialiseringsparametre.

4.5.6 config − den enkelte websides konfiguration

Objektet config (af type ServletConfig i pakken javax.servlet) giver adgang til konfiguration i web.xml for den enkelte JSP−side eller servlet.

String getInitParameter(String navn)

Giver værdien af initialiseringsparameteren *navn* defineret for websiden i web.xml.

Enumeration getInitParameterNames()

Giver en opremsning af navnene på alle initialiseringsparametrene for websiden.

Eksempel

```
<web−app>
  ..
  <servlet>
    <servlet−name>jspside</servlet−name>
    <jsp−file>/jspside.jsp</jsp−file>
    <init−param>
      <param−name>tekst</param−name>
      <param−value>Dette er en tekst fra web.xml</param−value>
    </init−param>
  </servlet>

</web−app>
```

Fra en JSP−side ville det kunne dette hentes med

```
config.getInitParameter("tekst")
```

som ville returnere strengen "Dette er en tekst fra web.xml".

Fra en servlet kunne teksten hentes med

```
    getInitParameter("tekst"));
```

4.5.7 page − selve JSP−siden

Dette objekt repræsenterer selve JSP−siden. Det er her taget med for en ordens skyld, selvom der ikke er meget man kan bruge det til i praksis.

4.5.8 exception − undtagelse opstået under kørsel

Objektet exception (af type Exception) repræsenterer en undtagelse der opstod under kørsel af en JSP−side. Objektet er kun defineret på fejlmeddelelsessider (hvor side−direktivet isErrorPage er sat til "true").

Se også afsnit 10.4.5. Et eksempel findes i afsnit 4.4.

4.5.9 pageContext − alle objekterne samlet i ét

Objektet pageContext (af type PageContext i pakken javax.servlet.jsp) giver adgang til alle de andre implicit definerede objekter. Det kan være nyttigt at bruge, hvis man ønsker at have alle de implicitte objekter samlet i ét objekt − f.eks. til et metodekald.

JspWriter getOut()
Giver out−objektet

HttpSession getSession()
Giver session−objektet

Object getPage()
Giver page−objektet

ServletRequest getRequest()
Giver request−objektet

ServletResponse getResponse()
Giver response−objektet

Exception getException()
Giver exception−objektet

ServletConfig getServletConfig()
Giver config−objektet

ServletContext getServletContext()
Giver application−objektet

4.6 Opgaver

Forbedring af ønskeseddel

1. Forbedr ønskeseddel−eksemplet i afsnit 4.1.1, sådan at brugeren også kan angive sit navn. Navnet huskes efterfølgende i session−objektet og skrives i sidens titel.
2. Lav mulighed for, at brugeren kan slette alle sine ønsker (navnet forbliver)
3. Lav mulighed for, at brugeren kan 'logge ud' (alle ønsker og navnet slettet).
   Prøv at bruge session.invalidate() (beskrevet i afsnit 4.5.4) for at smide sessionen væk.

Opgave: Afgiv din stemme

Lav et program, hvor man kan angive, hvem man vil stemme på til næste valg. Lad f.eks. brugeren vælge mellem 4 partier: Venstre, Konservative, Socialdemokratiet og SF.

Benyt radioknapper, sådan at man kun kan stemme på et parti ad gangen.
Optæl stemmerne og vis resultatet på skærmen hver gang brugeren har stemt.
Vink: Brug session− og application−objektet. Du kan søge inspiration ved at kigge i afsnit 3.3, Appendiks: Typer af formularfelter.

4.7 Test dig selv

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

4.8 Resumé

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

4.9 Avanceret: Fejlfinding i JSP

Det kan til tider være svært at finde og rette fejl i JSP−sider. Her følger et par idéer til, hvordan du kan lokalisere og rette fejlene.

4.9.1 Del og hersk

Har du en fejl, der driller, er det altid en god strategi, at prøve at isolere fejlen.

Ofte står man med en udgave af koden, der virker og en, der ikke virker. Prøv da 'del og hersk'−metoden: Find frem til fejlen ved langsomt at kopiere koden over (eller kommentere den ud og derpå kommentere den ind igen, bid for bid), sådan at du tilnærmer den kode, der virker, til den, der ikke virker.

4.9.2 Tjek om blokparenteser er balancerede

Som nævnt i afsnit 2.3.1, Blandet Java og HTML, skal man være meget omhyggelig med at omkranse HTML−kode i {− og }−parenteser og sørge for, at der altid er lige mange {−startparenteser som }−slutparenteser.

Er der rod med parenteserne får man typisk fejlmeddelelserne:

```
'try' without 'catch' or 'finally'
'catch' without 'try'
'else' without 'if'
```

4.9.3 Kigge på den oversatte servlet

JSP−siden bliver internt oversat til en servlet, der så oversættes igen, fra .java−kildetekst til en binær .class−fil (beskrevet i afsnit 7.3, Avanceret: JSP−siders interne virkemåde).

Det er ofte svært at finde fejl i en JSP−side, fordi de linjenumre, der angives i fejlmeddelelsen, normalt henviser til linjenumre i servlettens .java−kildetekst i stedet for de oprindelige linjenumre i JSP−siden.

Udviklingsmiljøer som (den kommercielle udgave af) Borland JBuilder, Sun One Studio o.s.v. tager højde for dette og finder de oprindelige linjenumre og tjekker også løbende JSP−sidernes syntaks, så man med det samme ser fejlen det rigtige sted.

De oversatte servletter ligger normalt i work/−mappen i webserveren og hedder nogenlunde det samme som JSP−siderne (beskrevet i afsnit 7.3).

4.9.4 Kigge i log−filerne

Når der opstår en fejl skrives en fejlmeddelelse i webserverens log. Det kan derfor være en god idé at kigge i disse.

Under Tomcat findes logfilerne i mappen logs/.

En snild måde under Linux til at kigge i alle logfilerne på én gang er fra kommandolinjen at skrive:

```
tail −f −n0 logs/*
```

Kommandoen 'tail' åbner herefter alle logfilerne og følger med i dem (−f), sådan at alle nye fejlmeddelelser skrives ud på skærmen.

Husk altid at udskrive fejlmeddelelserne og staksporene

Det er selvfølgelig afgørende for, om der er noget interessant at se i logfilerne, at dit program ikke undertrykker eventuelle fejl, men i stedet skriver dem ud.

Det er altså meget væsentligt at der ingen steder står:

```
try {
  ...
} catch (Exception e) { } // Forkert! Fejlen undertrykkes!
```

men i stedet, i alle try−catch−blokke, står:

```
try {
  ...
} catch (Exception e) { e.printStackTrace(); } // Udskriv stakspor i loggen
```

sådan at staksporet udskrives i loggen

4.9.5 Forstå staksporet

Et eksempel på et stakspor er vist nedenfor. Staksporet fortæller præcist hvad der skete, men det er kun meget få dele af det (herunder fremhævet med fed), der har vores interesse. De andre afspejler nemlig blot måden webserveren og klassebibliotekerne fungerer på:

```
java.sql.SQLException: General error,  message from server: "Table 'test.gaestebog' doesn't exist"
  com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:1825)
  com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1020)
  com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1109)
  com.mysql.jdbc.MysqlIO.sqlQuery(MysqlIO.java:1070)
  com.mysql.jdbc.Connection.execSQL(Connection.java:2027)
  com.mysql.jdbc.Connection.execSQL(Connection.java:1984)
  com.mysql.jdbc.Statement.executeQuery(Statement.java:1152)
  org.apache.jsp.JSP.kode.kapitel_05.gaestebog_jsp._jspService(gaestebog_jsp.java:58)
  org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:133)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:856)
  org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:311)
  org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:301)
  org.apache.jasper.servlet.JspServlet.service(JspServlet.java:248)
  javax.servlet.http.HttpServlet.service(HttpServlet.java:856)
```

Oversat til dansk siger fejlmeddelelsen at:

- Fejlen opstod i metoden checkErrorPacket(), på linje 1825 i filen MysqlIO.java.
- Denne metode blev kaldt fra sendCommand(), på linje 1020 i filen MysqlIO.java.
- ... (flere metoder der var i gang med at kalde hinanden) ...
- **executeQuery() blev kaldt i metoden _jspService() på linje 58 i gaestebog_jsp.java**
- ... (flere metoder der var i gang med at blive kaldt) ...

Som regel fortæller staksporet alene nok til at man kan finde fejlen. Kigger man på gaestebog.jsp (den kommer i afsnit 5.5, Eksempel − gæstebog) ser man at executeQuery() kun kaldes ét sted, og man har således fundet frem til stedet, hvor fejlen opstod.

Et man stadig i tvivl om, præcis hvor fejlen opstod (måske er der flere steder med kald til executeQuery()), må man bruge sin viden om, at det skete i linje 58 i gaestebog_jsp.java og kigge på den oversatte servlet (ligger normalt i work/−mappen i webserveren, se afsnit 7.3.1, Kigge i de genererede servletter).

4.9.6 Hvis klasse(bibliotek)er ikke kan findes

Får du fejlen NoClassDefFoundError, er det et symptom på, at en klasse ikke kan findes. Det kan f.eks. ske, når du prøver at indlæse en databasedriver (der kan du også få 'No suitable driver') eller, hvis du bruger ekstra klassebiblioteker ud over de, der er defineret i standard Java JDK.

Ekstra klassebiblioteker og drivere fås næsten altid som JAR−filer (der er et ZIP−arkiv med en række .class−filer).

Der er mange muligheder for at løse problemet med, at en klasse ikke kan findes:

- Hvis du arbejder i et udviklingsværktøj (f.eks. for at kunne oversætte nogle klasser/JSP−sider der bruger klasserne): Inkludér de rigtige biblioteker (eng.: libraries) med de JAR−filer, der indeholder klasserne (opret evt. et nyt bibliotek), i dit projekt.
- Hvis programmet startes selvstændigt, evt. fra kommandolinjen: Sæt en henvisning til JAR−filen ind i miljøvariablen CLASSPATH
- Hvis klasserne skal være kendt i ALLE Java−programmer (der kører på en bestemt JDK): Kopier JAR−filerne over i JDK−et, under jre/lib/ext/−mappen
- Hvis klasserne skal være kendt i ALLE webapplikationer (der kører i en bestemt Tomcat webserver): Kopier JAR−filerne over i Tomcats common/lib/−mappe
- Hvis klasserne skal være kendt i en webapplikation, sådan at denne webapplikation også kunne flyttes over på en anden server: Kopiér JAR−filerne over i webapplikationens WEB−INF/lib/−mappe (opret evt mappen − se også afsnit 7.4).

I princippet kan alle mulighederne være lige gode til at løse problemet, men når du har fået det til at virke, så brug lidt tid på at overveje om løsningen er den rigtige for dig (det er f.eks. lidt kedeligt hvis du ofte skifter JDK og du netop har lagt databaseklasserne ind i dit JDK).

1Omdirigering skal ske så tidligt på siden, at man kan være sikker på, at der ikke er genereret så meget HTML, at bufferen er løbet fuld og HTTP−hovedet (se afsnit 3.7) og den første del af data således er sendt til klienten. Når først data er sendt til klienten, kan webserveren ikke 'fortryde' og i stedet omdirigere til en anden side.
For servletter er der ingen output−buffer, så i servletter skal omdirigering ske inden der har været noget output overhovedet.

2Som vi senere skal se, i afsnit 7.3, Avanceret: JSP−siders interne virkemåde, bliver JSP−sider lavet om til en metode i en klasse. Man kan undgå at den efterfølgende kode i metoden udføres ved at returnere fra metoden, med 'return;'. I afsnit 10.4.4 er der et eksempel på dette.

3Denne opførsel kan slås fra ved at sætte session="false" i sidedirektivet med
<%@ page session="false" %>

4En mindre hensigstmæssig måde kunne være at bruge en klassevariabel i en separat klasse.

5 Brug af databaser

En overordnet forståelse af databaser forudsættes i store dele af resten af bogen. Kapitlet forudsætter kapitel 3, Interaktive sider og lidt kendskab til databaser.

5.1 Strategier til databaseadgang

I dette kapitel vil vi se på, hvordan man kommunikerer med en database fra Java og JSP.

Der er rigtig mange muligheder for, hvordan databasekommunikationen kan foregå. De fleste muligheder involverer JDBC (Java DataBase Connectivity), som bliver gennemgået i dette kapitel.

Nogle af mulighederne er:

- At bruge JDBC direkte fra JSP−siderne (som beskrevet i f.eks. dette kapitel i afsnit 5.5, Eksempel − gæstebog)
- At bruge JDBC fra Java−klasser, der så bruges fra JSP−siderne (beskrevet i det følgende, se kapitel 9, Javabønner i JSP−sider, hvordan klasser kan bruges fra JSP−sider)
- At bruge JSTLs databasefunktioner fra JSP (se afsnit 6.3, JSTL og databaser (<sql: >))
- At bruge JDBC RowSet (se afsnit 5.9, JDBC RowSet), herunder:
- Bruge CachedRowSet direkte fra JSP−sider (eller Java−klasser)
- Bruge WebRowSet til at producere XML, der derpå kan transformeres
- med XSL (XML Style Sheet − ikke beskrevet i denne bog, men se afsnit 11.1.4)
- med JSTLs XML−tags (se et eksempel i afsnit 6.4.3)
- At bruge EJB − Enterprise JavaBeans (se kapitel 12) med containerstyret persistens.
- At bruge JDO (Java Data Objects − ikke beskrevet i denne bog)

Hvilken løsning der er bedst afhænger af, hvor omfattende ens webapplikation er og hvor meget man forventer den senere skal vedligeholdes. Dette vil blive diskuteret nærmere i afsnit 10.2, Model 1 og model 2−arkitekturer.

Er du begynder og skal til at lave din første webapplikation, anbefales det, at du bruger en af de to første muligheder (JDBC enten fra JSP eller Java−klasser). Du kan så kigge på de andre muligheder på et senere tidspunkt.

5.1.1 Hvilken database skal man vælge

Det første skridt for at komme i gang med databaser er naturligvis, at installere en database, man kan kommunikere med.

Bruger du Windows og har du Microsoft Office−pakken installeret, kan du vælge at bruge Microsoft Access−databasen. Den er dog ikke særlig velegnet til større projekter. Har du adgang til en anden kommerciel database, såsom Oracle, kan du med fordel bruge den.

Her i bogen vil vi bruge MySQL, som er en meget populær database med Åben Kildekode (eng.: Open Source), der er gratis tilgængelig fra http://mysql.com.

Under Linux er den højst sandsynligt med i din distribution, så du skal blot aktivere den i kontrolpanelet, så vil den automatisk blive installeret.

Bruger du et andet styresystem (f.eks. Windows eller Mac) må du selv hente den ned fra http://mysql.com og installere den.

5.2 I gang med databaser og MySQL

Når man kommunikerer med en database, bruger man som regel sproget SQL (Structured Query Language).

5.2.1 SQL−kommandoer og forespørgsler

Det følgende viser nogle eksempler på SQL−sproget.

For at oprette en tabel kaldet kunder med to kolonner, et navn som er en tekst og en kredit som er et tal, skriver man:

```
CREATE TABLE kunder (navn varchar(32), kredit float);
```

Tabellen er tom til at starte med. For at tilføje to kunder skriver man:

```
INSERT INTO kunder VALUES('Jacob', −1799);
INSERT INTO kunder(navn,kredit) VALUES('Brian', 0);
```

Nu er der to rækker (poster) i tabellen. Da vi tilføjede 'Brian' skrev vi også kolonnenavnene (hvis man ikke kender rækkefølgen eller der måske er flere kolonner er det en god idé).

Nu ville tabellen se ud som følger:

navn	kredit
Jacob	−1799
Brian	0

Hele tabellen kunne udskrives med SQL−forespørgslen

```
SELECT * FROM kunder;
```

Denne forespørgsel kunne forfines, f.eks. til at fremsøge navne, hvor kreditten er negativ:

```
SELECT navn FROM kunder WHERE kredit > 0;
```

Der som svar blot giver 'Jacob'.

Man kan naturligvis også opdatere tabellen. Lad os rette i Brians kredit:

```
UPDATE kunder SET kredit=10 WHERE navn='Brian';
```

Man kan slette. Her sletter f.eks. vi rækken (eller rækkerne) hvor navnet er Jacob:

```
DELETE FROM kunder WHERE navn='Jacob';
```

Vi kunne også slette hele tabellen med:

```
DROP TABLE kunder;
```

Her er alle kommandoerne samlet:

```
-- Filnavn: sqleksempler.sql
-- Kan bruges fra kommandolinjen med: mysql test < sqleksempler.sql

CREATE TABLE kunder (navn varchar(32), kredit float); -- opret tabel
INSERT INTO kunder VALUES('Jacob', -1799);          -- indsæt række
INSERT INTO kunder(navn,kredit) VALUES('Brian', 0); -- do, med navngivne kolonner
UPDATE kunder SET kredit=10 WHERE navn='Brian';     -- sæt Brians kredit til 10 kr
DELETE FROM kunder WHERE navn='Jacob';              -- slet række om Jacob
SELECT * FROM kunder;                               -- vis hele tabellen
DROP TABLE kunder;                                  -- slet tabellen igen
```

5.2.2 Administrere databasen fra kommandolinjen

Efter at MySQL er blevet installeret, skal den startes op. Det kan enten gøres ved at installere den som en tjeneste, der startes automatisk, når computeres starter op eller ved at starte programmet 'mysqld' manuelt (som superbruger).

Når MySQL installeres, opretter den som standard en tom database med navnet 'test', der kan tilgås af alle brugere.

Somme tider kan det være nødvendigt at arbejde med databasen fra kommandolinjen, måske fordi man ikke har adgang til en grafisk brugergrænseflade.

Man kan logge på test−databasen med kommandoen

```
  mysql test
```

Her er et fuldt eksempel på, hvordan kommunikation med MySQL fra kommandolinjen kunne se ud. Det brugeren skriver er i fed:

```
$> mysql test
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 4.0.15

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE TABLE kunder (navn varchar(32), kredit float);
Query OK, 0 rows affected (0.32 sec)

mysql> INSERT INTO kunder VALUES('Jacob', -1799);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO kunder(navn,kredit) VALUES('Brian', 0);
Query OK, 1 row affected (0.00 sec)

mysql> UPDATE kunder SET kredit=10 WHERE navn='Brian';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM kunder;
+-------+--------+
| navn  | kredit |
+-------+--------+
| Jacob |  -1799 |
| Brian |     10 |
+-------+--------+
2 rows in set (0.00 sec)

mysql> show tables;
+-----------------+
| Tables_in_test  |
+-----------------+
| kunder          |
+-----------------+
1 rows in set (0.00 sec)

mysql> exit
Bye
```

Sammenlign med afsnit 5.2.1, SQL−kommandoer og forespørgsler. Se også MySQL−manualen (http://mysql.com/doc/ under 'Tutorial') for en introduktion til SQL og MySQL.

Brugere i databasen

Som udgangspunkt findes databasebrugeren " (intet brugernavn) og brugeren 'root' i MySQL. Begge brugere kan kun logge på fra den lokale maskine uden adgangskode.

Ovenfor har vi brugt brugeren med et tomt brugernavn, der kun kan se databasen 'test' og ikke kan oprette nye databaser.

Ønsker man at bruge en anden databasebruger og/eller adgangskode skriver man:

```
mysql -u brugernavn -p
```

og indtaster adgangskoden (-p angiver at adgangskoden skal læses fra tastaturet). For en bruger uden adgangskode, f.eks. bruger 'root' skriver man:

```
mysql -u root
```

Oprette en database

Ønsker man at oprette en anden database kan det gøres fra kommandolinjen med:

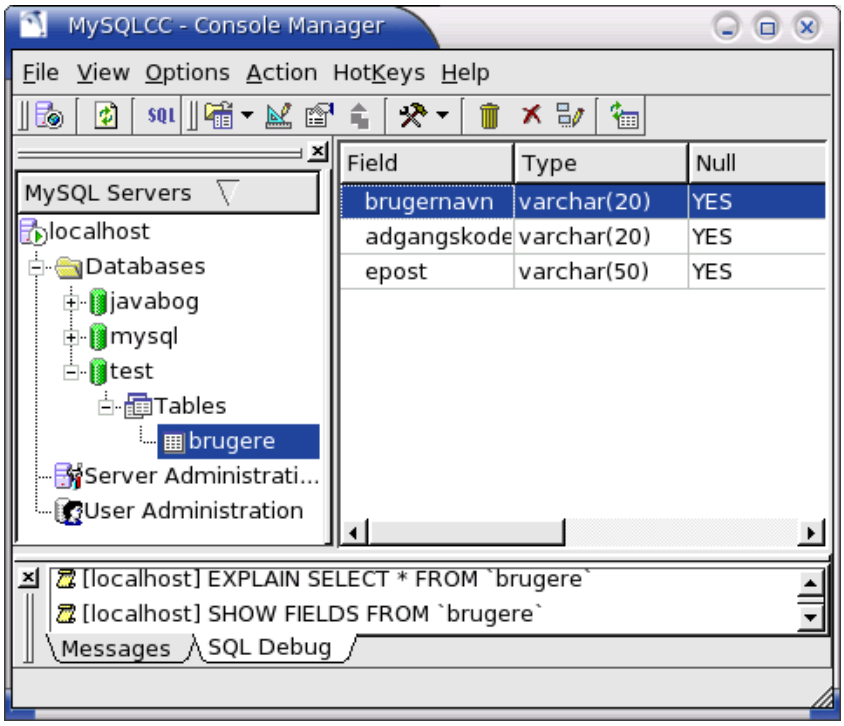
```
mysqladmin -u root create jspdb
```

Nu er databasen 'jspdb' oprettet. Derefter kan man gå ind i den med:

```
mysql -u root jspdb
```

5.2.3 Administrere databasen med MySQLCC

Fra MySQL.com kan man også hente MySQL Control Center (MySQLCC), som kan hjælpe med at administrere ens database, herunder oprette og ændre i databaser og tabeller samt selvfølgelig indsætte, slette og søge i tabellerne:



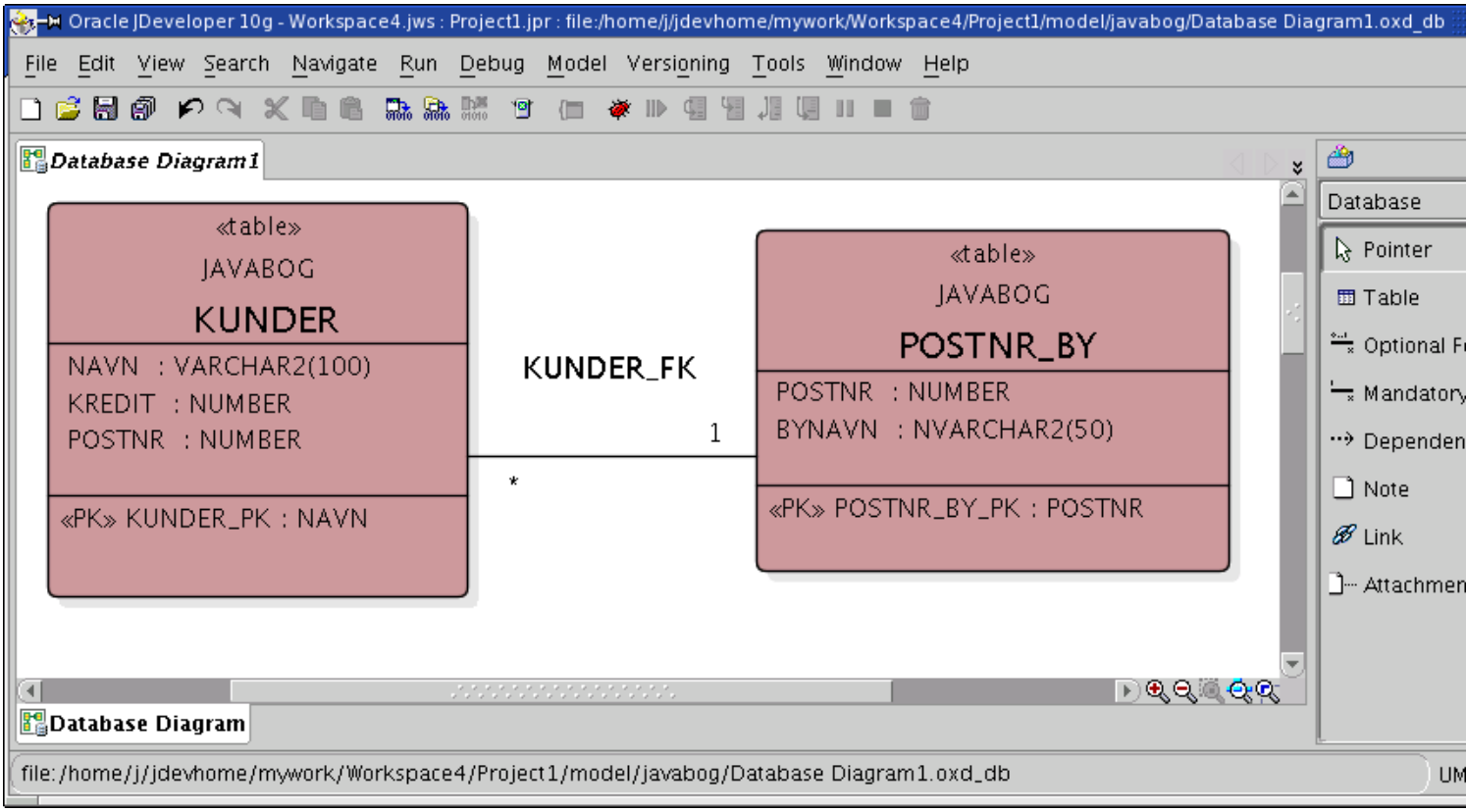
MySQLCC findes til Linux og Windows. Herover er det vist under Linux.

Værktøjet er meget pædagogisk, idet man kan vælge at få vist præcist hvilke SQL-kommandoer, der udføres (nederst på billedet). Dette kan man så bruge til selv at lære SQL.

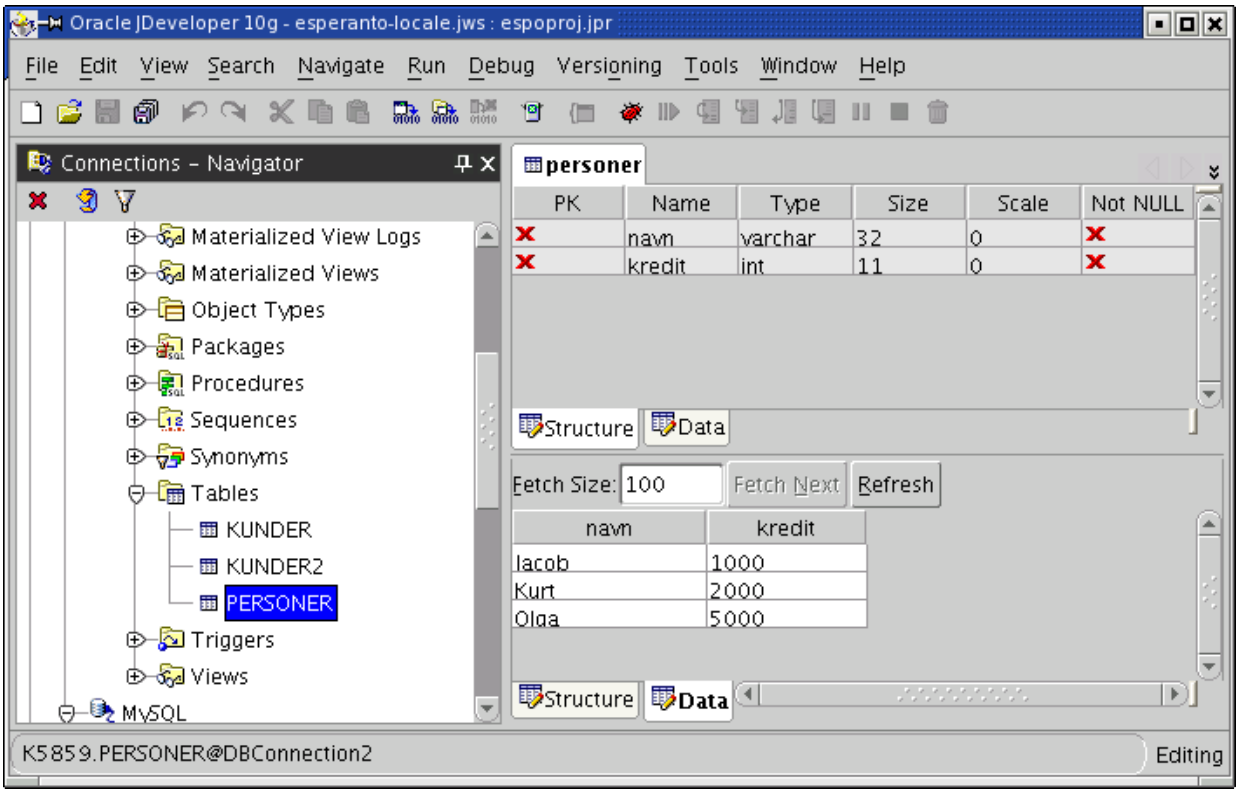
5.2.4 Administrere fra et udviklingsværktøj

De fleste udviklingsværktøjer har mere eller mindre sofistikerede SQL-værktøjer.

En af de mest raffinerede er Oracle JDeveloper, der lader en designe tabellerne visuelt:



... og senere hen vise (og, hvis det er en Oracle−database, også redigere) tabellen:



5.3 Kontakt til database fra Java (JDBC)

Adgang til en database fra Java sker gennem et sæt klasser, der under et kaldes JDBC (Java DataBase Connectivity). Disse klasser ligger i pakken java.sql.

JSP−sider, der arbejder med databaser skal derfor importere denne pakke i starten:

```
<%@ page language="java" import="java.sql.*" %>
```

Tilsvarende skal der i starten af Java−filer, der arbejder med databaser, stå:

```
import java.sql.*;
```

At få kontakt til en database fra Java kræver to skridt:

1. Indlæse databasedriveren
2. Etablere forbindelsen

Indlæsning af driveren sker, ved at bede systemet indlæse den pågældende klasse, der derefter registrerer sig selv i JDBC−systemets driver−manager.

Ofte skal man have en JAR−fil (et Java−ARkiv, en samling klasser pakket i ZIP−formatet) med en driver fra producenten. På http://java.sun.com/jdbc kan man finde over 200 drivere til forskellige databaser.

5.3.1 Kontakt gennem ODBC under Windows

Med Java under Windows følger en standard JDBC−ODBC−bro med, så man kan kontakte alle datakilder defineret under ODBC. Denne driver indlæses med:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Når forbindelsen oprettes, angiver man den ønskede datakildes navn.

```
Connection con = DriverManager.getConnection("jdbc:odbc:datakilde1");
```

Husk, at datakildens navn (her "datakilde1") først skal være defineret i Windows' Kontrolpanel under ODBC. Datakilden kan være en fil, f.eks. en Microsoft Access−fil (.mdb) eller en rigtig database.

Bemærk, at ODBC er en ret langsom protokol, der ikke understøtter ret mange af de mere avancerede ting. Har du brug for bedre ydelse bør du finde en driver, der kommunikerer direkte med databasen/filen, i stedet for at bruge JDBC−ODBC.

5.3.2 Kontakt til MySQL−database

På http://mysql.com kan hentes en JDBC−driver til MySQL (kaldet Connector/J).

Driver−filen hedder typisk noget a la mysql−connector−java−3.0.9−bin.jar. Den skal ligge i CLASSPATH eller kopieres over i mappen java/jre/lib/ext/ så Java kender til driveren[1].

I kildeteksten skriver man:

```
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection("jdbc:mysql:///test");
```

hvorefter man skulle have kontakt til databasen 'test' på den lokale maskine (følger som standard med i MySQL), som anonym bruger.

Ønsker man at bruge en anden database, som en anden bruger, skriver man f.eks.:

```
Connection con = DriverManager.getConnection("jdbc:mysql:///jspdb","j","hemli");
```

hvorefter man skulle have oprettet forbindelsen til databasen 'jspdb' på den lokale maskine med brugernavn 'j' og adgangskode 'hemli'.

5.3.3 Kontakt til en Oracle−database

Ønsker man kontakt til en Oracle−database skriver man:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Driver−filen hedder classes12.zip og passer til en bestemt udgave af Oracle−databasen. Den skal ligge i CLASSPATH eller kopieres over i mappen java/jre/lib/ext/ så Java kender driveren.

Herefter kan man oprette forbindelsen med f.eks.:

```
Connection con = DriverManager.getConnection(
  "jdbc:oracle:thin:@ora.javabog.dk:1521:student","j","hemli");
```

Første parameter er en URL til databasen. Den består af en protokol (jdbc), underprotokol (oracle) og noget mere, der afhænger af underprotokollen. I dette tilfælde angiver det, at databasen ligger på maskinen ora.javabog.dk port 1521 og hedder student. Anden og tredje parameter er brugernavn og adgangskode.

5.4 Kommunikation med database fra Java

Når vi har en forbindelse, kan vi oprette et "statement"−objekt, som vi kan sende kommandoer og forespørgsler til databasen med:

```
Statement stmt = con.createStatement();
```

Der kan opstå forskellige undtagelser af typen SQLException, der skal håndteres.

For at opnå 'hul igennem' er det ofte en god idé at tage ét skridt ad gangen. Eksemplerne i det følgende er derfor skrevet som almindelige Java−klasser med en main()−metode, sådan at man kan prøve dem fra kommandolinjen eller i et udviklingsværktøj, uden for webserveren.

På http://java.sun.com/docs/books/tutorial/jdbc/ kan du få yderligere information om JDBC og databasekommunikation fra Java.

5.4.1 SQL−kommandoer og forespørgsler fra Java

SQL−kommandoer, der ikke giver et svar tilbage i form af data, såsom INSERT, UPDATE, DELETE, CREATE TABLE og DROP TABLE, sendes med executeUpdate()−metoden.

Her opretter vi f.eks. tabellen "kunder" og indsætter et par rækker med en database defineret som 'datakilde1' i Windows' ODBC−kontrolpanel.

```
import java.sql.*;
public class TestDatabaseforbindelseODBC
{
  public static void main(String[] arg) throws Exception
  {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection con = DriverManager.getConnection("jdbc:odbc:datakilde1");
    Statement stmt = con.createStatement();

    stmt.executeUpdate("CREATE TABLE kunder (navn varchar(32), kredit float)" );
    stmt.executeUpdate("INSERT INTO kunder VALUES('Jacob', −1799)");
    stmt.executeUpdate("INSERT INTO kunder (navn,kredit) VALUES('Brian', 0)");
  }
}
```

Tilsvarende med en MySQL−database:

```
import java.sql.*;
public class TestDatabaseforbindelseMySQL
{
  public static void main(String[] arg) throws Exception
  {
    Class.forName("com.mysql.jdbc.Driver");
    Connection con = DriverManager.getConnection("jdbc:mysql:///test");
    Statement stmt = con.createStatement();

    stmt.executeUpdate("CREATE TABLE kunder (navn varchar(32), kredit float)" );
    stmt.executeUpdate("INSERT INTO kunder VALUES('Jacob', −1799)");
    stmt.executeUpdate("INSERT INTO kunder(navn,kredit) VALUES('Brian', 0)");
  }
}
```

Forespørgsler

SQL−forespørgslen SELECT udføres med metoden executeQuery():

```
  ResultSet rs;
  rs = stmt.executeQuery("SELECT navn, kredit FROM kunder WHERE navn='Jacob'");
```

Den giver et ResultSet−objekt, der repræsenterer svaret på forespørgslen (for at få alle kolonner kunne vi også skrive "SELECT * FROM kunder").

Data hentes fra ResultSet−objektet således:

```
  while (rs.next())
  {
    String navn = rs.getString("navn");
    double kredit = rs.getDouble("kredit");
    System.out.println(navn+" "+kredit);
  }
```

Man kalder altså next() for at få næste række i svaret, læser de enkelte celler ud fra kolonnenavnene (eller kolonnenumrene, regnet fra 1 af), hvorefter man går videre til næste række med next() osv. Når next() returnerer false, er der ikke flere rækker at læse.

SQL med data fra variabler

Ofte er data gemt i variabler og kommer fra brugeren f.eks. fra request−objektet:

```
    String navn = request.getParameter("navn");
```

Så må man sætte en streng sammen, der giver den ønskede SQL−kommando:

```
    int kredit = 500;

    // indsæt data fra variablerne navn og kredit
    stmt.executeUpdate("INSERT INTO kunder VALUES('"+navn+"', "+kredit+")");
```

Her ville den resulterende streng blive (hvis variablen navn indeholdt værdien "Hans") "INSERT INTO kunder VALUES('Hans', 500)" og databasen ville indsætte denne række.

På fuldstændig samme måde som med SQL−kommandoer, kan SQL−forespørgsler laves, ved at sætte en passende SQL−streng sammen ud fra en variabel:

```
  rs = stmt.executeQuery("SELECT navn, kredit FROM kunder WHERE navn='"+navn+"'");
```

Vær dog opmærksom på ikke, at introducere nogle sikkerhedsproblemer. I længden er det bedre at bruge på forhånd forberedt SQL, i stedet for at stykke en streng sammen med SQL−kommandoen.

5.4.2 På forhånd forberedt SQL

I koden vist ovenfor er der nogle problemer:

1. Ved hvert kald bruges noget tid på at sætte en SQL−streng sammen, som databasen derefter skal fortolke for at udføre SQL−kommandoen.
2. Hvis navnet indeholder et ' går SQL−kommandoen galt. F.eks. hvis navn havde værdien "Hans' venner" ville SQL−strengen blive:
   `"INSERT INTO kunder VALUES('Hans' venner', 500)"`
   som er ugyldig fordi den indeholder tre '−er.
3. Endnu værre er det med sikkerheden. Det er nemlig muligt at *injicere SQL* (eng.: SQL injection): Hvis brugeren er så smart, at han angiver et meget specielt navn, f.eks. navn="Hans', 1000000) −−". Da vil SQL−strengen blive:
   `"INSERT INTO kunder VALUES('Hans', 1000000) --', 500)"`
   som er gyldigt SQL (−− angiver at resten af linjen er en kommentar). På den måde kunne brugeren oprette sig selv i databasen med vilkårligt store summer i kredit!
   Se afsnit 8.5 for en generel diskussion af sikkerhed.

I stedet for at oprette en SQL−kommandolinje med createStatement(), bør man derfor bruge metoden prepareStatement(), hvor man angiver SQL−kommandoen én gang og derefter kan udføre kommandoen med forskellige data mange gange.

I SQL−kommandoen skrives så et ?−tegn for hver stump data, der kan variere. Før hvert kald sættes data med en set−metode og nummeret på stumpen (regnet fra og med 1):

```
import java.sql.*;
public class ForberedtSQL
{
  public static void main(String[] arg) throws Exception
  {
    Class.forName("com.mysql.jdbc.Driver");
    Connection con = DriverManager.getConnection("jdbc:mysql:///test");

    // Forbered kommandoerne til databasen, f.eks. i starten af programmet:

    PreparedStatement indsætPstm = con.prepareStatement(
                               "INSERT INTO kunder (navn,kredit) VALUES(?, ?)");

    PreparedStatement hentPstm = con.prepareStatement(
                               "SELECT navn, kredit FROM kunder WHERE navn=?");

    // under programudførelsen kan de forberedte kommandoer udføres mange gange:
    for (int i=0; i<100; i++)
    {
      indsætPstm.setString(1, "Brian");
      indsætPstm.setInt(2, i);
      indsætPstm.execute();

      indsætPstm.setString(1, "Hans' venner");  // bemærk ' i strengen
      indsætPstm.setInt(2, 1042+i);
      indsætPstm.execute();


      hentPstm.setString(1, "Hans' venner");    // bemærk ' i SQL−forespørgslen
      ResultSet rs = hentPstm.executeQuery();

      // man løber igennem svaret som man plejer
      while (rs.next())
      {
        String navn = rs.getString(1);
        double kredit = rs.getDouble(2);
        System.out.println(navn+" "+kredit);
      }
    }
  }
}
```

Øvelse

Skriv ovenstående program om, til at bruge et almindeligt statement−objekt. Hvor stor er forskellen i udførelseshastighed (fjern kaldet til System.out.println() for at få en klar måling)?

Om optimering

Ovenstående forbedrer ganske vist ydelsen en del (afhængig af JDBC−driveren), men det vigtigste er, at ' i strengene håndteres, den øgede sikkerhed og (efter forfatterens smag) den øgede overskuelighed.

Læs mere om optimering i afsnit 5.8.

5.5 Eksempel − gæstebog

Et eksempel på databasekommunikation kunne være en gæstebog, hvor brugerne kan se alle andre brugere og hvad de har skrevet ind.

5.5.1 Oprette tabellen i databasen

Først skal tabellen i databasen oprettes. Hver række har en unik id, navnet på brugeren, beskedens tekst og datoen hvor beskeden blev oprettet. IP−adressen bliver af sikkerhedshensyn også noteret. Her er SQL−koden til at oprette tabellen:

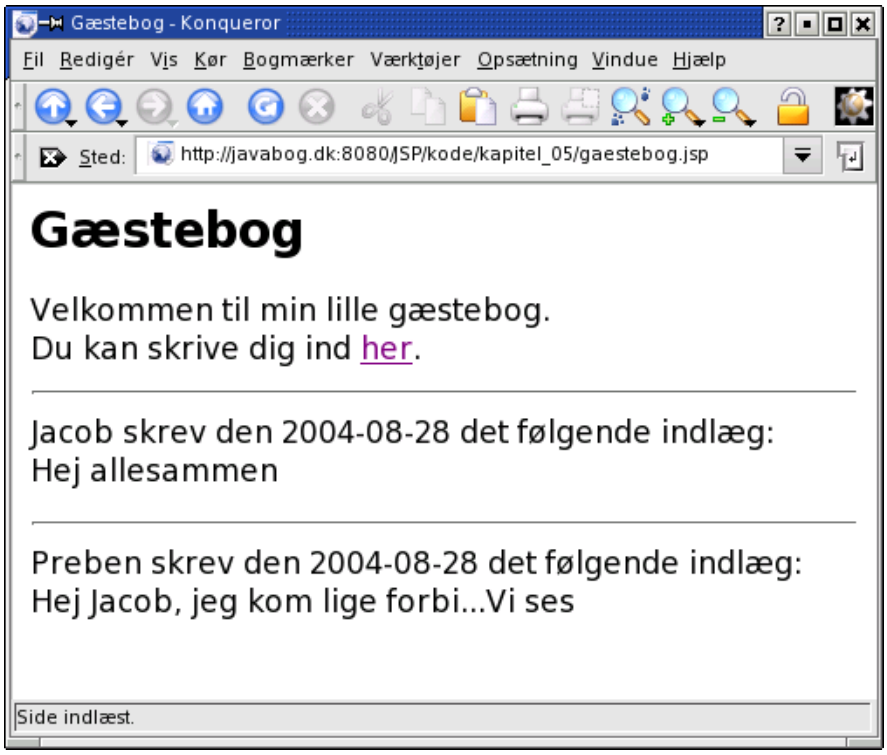
```
-- Filnavn: gaestebogsoprettelse.sql
-- Kan bruges fra kommandolinjen med: mysql test < gaestebogsoprettelse.sql

CREATE TABLE gaestebog (
  id      int  not null  primary key  auto_increment,
  navn    varchar(20),
  besked  varchar(255),
  dato    datetime,
  ip      varchar(16)
);
```

Denne SQL kan udføres som beskrevet i afsnit 5.2.2, Administrere databasen fra kommandolinjen, eller man kan lave et lille program, der opretter tabellen:

```
import java.sql.*;
public class Gaestebogsoprettelse
{
  public static void main(String[] arg) throws Exception
  {
    Class.forName("com.mysql.jdbc.Driver");
    Connection con = DriverManager.getConnection("jdbc:mysql:///test");

    Statement stmt = con.createStatement();

    stmt.executeUpdate(
      "CREATE TABLE gaestebog ("
      +"id      int  not null  primary key  auto_increment,"
      +"navn    varchar(20),"
      +"besked  varchar(255),"
      +"dato    datetime),"
      +"ip      varchar(16)");
  }
}
```

5.5.2 Visning af gæstebogen (gaestebog.jsp)

Her er siden, der viser gæstebogen:



gaestebog.jsp

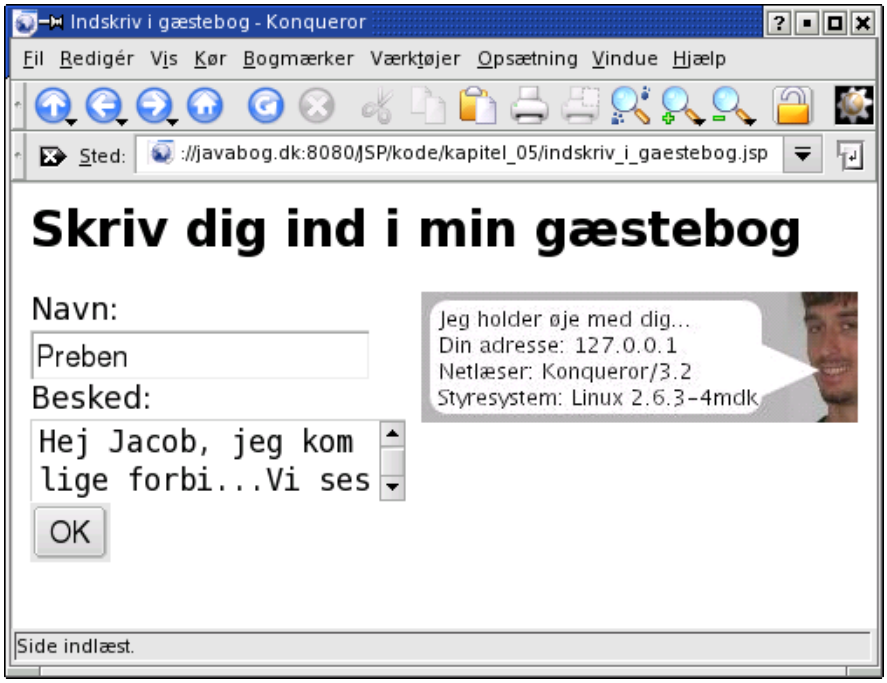
```
<%@ page language="java" import="java.sql.*" %>
<html>
<head><title>Gæstebog</title></head>
<body>

<h1>Gæstebog</h1>
Velkommen til min lille gæstebog.<br>
Du kan skrive dig ind <a href="indskriv_i_gaestebog.jsp">her</a>.

<%
  // Udskriv gæstebogen
  Class.forName("com.mysql.jdbc.Driver");
  Connection con = DriverManager.getConnection("jdbc:mysql:///test");
  Statement stmt = con.createStatement();
  ResultSet rs = stmt.executeQuery("SELECT navn, besked, dato FROM gaestebog");

  while(rs.next()) {
    %>
    <hr>
      <%= rs.getString("navn") %> skrev den <%= rs.getDate("dato") %>
      det følgende indlæg:<br>
      <%= rs.getString("besked") %>
    <p>
    <%
  } // slut på while-løkke

  // oprydning
  rs.close();
  stmt.close();
  con.close();
%>
</body>
</html>
```

5.5.3 Indskrivning (indskriv_i_gaestebog.jsp)

Vælger brugeren at indskrive sig, kommer han til følgende side:

indskriv_i_gaestebog.jsp

```
<%@ page language="java" import="java.sql.*" %>
<html>
<head><title>Indskriv i gæstebog</title></head>
<body>

<h1>Skriv dig ind i min gæstebog</h1>

<form>           <img src="../kapitel_02/billede_med_avatar.jsp" align="right">
Navn: <br>
<input type="text" name="navn" size=20><br>
Besked: <br>
<textarea name="besked" width="50" height="10"></textarea><br>
<input type="submit" value="OK">
</form>

<%
  // Se om der kommer nogen data fra formularen
  String navn = request.getParameter("navn");
  String besked = request.getParameter("besked");

  if (navn != null && besked != null)
  {
    // OK, vi har fået en indtastning. Tjek om brugeren reelt har tastet noget.
    if (navn.length() < 3 || besked.length() < 3)
    {
      out.write("Indtast venligst dit fulde navn og en besked.");
    }
    else
    {
      // OK, beskedden er gyldig. Indsæt den i databasen
      Class.forName("com.mysql.jdbc.Driver");
      Connection con = DriverManager.getConnection("jdbc:mysql:///test");

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

      // alt gik godt. Send brugeren tilbage til gæstebogen
      response.sendRedirect("gaestebog.jsp");
    }
  }
%>
</body>
</html>
```



5.6 Test dig selv

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

5.7 Resumé

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

5.8 Avanceret: Optimering

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

5.8.1 Bruge den rigtige databasedriver

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

5.8.2 På forhånd forberedt SQL

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

5.8.3 Lægge opdateringer i kø (batch)

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

5.8.4 Lægge 'stored procedures' i databasen

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

5.8.5 Forbindelsespuljer (Connection pooling)

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

5.8.6 Metadata

5.8.7 Eksempel: Webgrænseflade til database

5.9 Avanceret: JDBC RowSet

5.9.1 JdbcRowSet

5.9.2 CachedRowSet

5.9.3 FilteredRowSet og JoinRowSet

5.9.4 WebRowSet

5.9.5 Mere information

1Se afsnit 4.9.6 hvis du har problemer med at få webserveren til at finde JAR−filerne.

6 JSTL − JSP Standard Tag Library

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

Det forudsætter kapitel 4, Videre med JSP, men inddrager eksempler fra mange kapitler både før og efter dette kapitel. Det anbefales derfor at du springer let hen over det du ikke forstår og vender tilbage senere.

Med JSP 2.0, der udkom primo 2004, blev det for alvor muligt, at programmere JSP−sider i programmeringssproget JSTL (JSP Standard Tag Library) og bruge regneudtryk i disse sider (EL − Expression Language).

JSTL 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].

Selvom denne bogs 'hovedsprog' er Java og ikke JSTL, kan du vælge at lære JSTL. Dette kan du gøre ved at læse de andre kapitler og i stedet for at udføre Java−eksemplerne, kigge på og lege med JSTL−eksemplerne i dette kapitel.

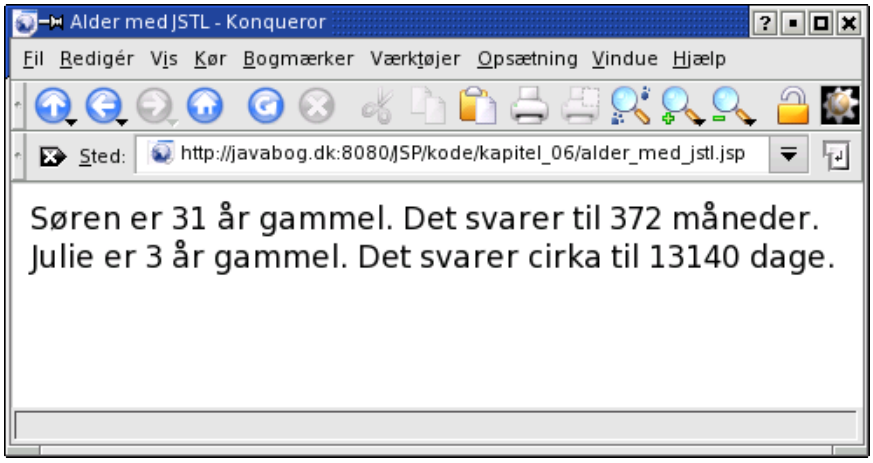
Et tag library (forkortet taglib) er, som navnet siger, et bibliotek af HTML−lignende koder. Ligesom JSP−koderne udføres taglib−koderne på serveren.

6.1 Kernefunktionalitet (<c: >)

Kernefunktioner (de mest basale funktioner) i JSTL ligger i core−taglibbet under navnet 'c'. Her er understøttelse for variabler, betingelser, URL−styring og andre ting.

Her er eksemplet Alder fra afsnit 2.2 skrevet i JSTL−syntaks:

```
<!-- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%> -->
<html>
<head><title>Alder med JSTL</title></head>
<body>
<p>
<c:set var="alder" value="31" />
Søren er ${alder} år gammel.
Det svarer til ${12*alder} måneder.<br>

<c:set var="alder" value="3" />
Julie er ${alder} år gammel.

<c:set var="alder" value="${alder*365}" />
Det svarer cirka til ${12*alder} dage.<br>
</p>
</body>
</html>
```



Vi starter med at importere core−taglibbet:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

I eksemplet ovenfor har vi sat det ind i en HTML−kommentar <!−− −−>. Det er strengt taget ikke nødvendigt, men sørger for, at de fleste HTML−redigeringsværktøjer (som f.eks. Netscape/Mozilla Composer) lader koden stå urørt, selvom de ikke står i JSP−kode. Almindelig javakode mellem <% og %> går ellers ofte går tabt.

Nu kan vi bruge core−biblioteket under præfikset "c" i JSP−siden, f.eks. til at sætte en variabel til en værdi:

```
<c:set var="alder" value="31" />
```

I modsætning til Java behøver man ikke definere typen af en variabel, før man bruger den.

6.1.1 Regneudtryk med EL − Expression Language

Værdier fra variabler og regneudtryk skal i JSTL puttes ind i et ${}−udtryk, som er en måde at gøre brug af EL − Expression Language. Med JSP 2.0 kan evaluering af ${}−udtryk ske uden for JSTL, sådan at de bliver evalueret, når de står direkte i HTML−koden:

```
Det svarer til ${12*alder} måneder.<br>
```

I tidligere udgaver kunne EL−udtryk kun stå inde i JSTL−koder, så man måtte skrive dem ind i f.eks. en <c:out />:

```
Det svarer til <c:out value="${12*alder}" /> måneder.<br>
```

Der kan stadig nogle enkelte steder være grund til at bruge <c:out />, nemlig hvis man ønsker at undgå, at værdien kan fortolkes som HTML−kode (se afsnit 8.5.1, HTML−injektion og afsnit 6.1.4, Sikkerhed og behandling af formularer).

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:

```
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).

6.1.2 Variabler

Variabler kan sættes med:

```
<c:set var="fornavn" value="Jacob"/>
```

der sætter variablen 'fornavn' til "Jacob". Findes variablen ikke, oprettes den.

En anden mulighed er, at sætte værdien ind i :

```
<c:set var="fornavn">
  Jacob
</c:set>
```

6.1.3 Virkefelter

Som udgangspunkt har variabler kun siden som virkefelt (scope="page"), så de smides væk, når forespørgslen er færdigbehandlet. Man kan specificere et andet virkefelt med f.eks.:

```
<c:set var="fornavn" value="Jacob" scope="session"/>
```

hvorefter variablen vil blive husket, så længe brugerens session eksisterer.

De mulige virkefelter er: page, request, session og application. Et (realistisk) eksempel på brug af virkefelter kan ses i slutningen af afsnit 6.4.2.

Vil man fjerne en variabel, kan det gøres med f.eks.:

```
<c:remove var="navn" scope="session"/>
```

6.1.4 Sikkerhed og behandling af formularer

JSTL har med EL nogle ganske kraftige mekanismer til at arbejde med formulardata.

Her eksemplet fra afsnit 3.2.5 (der udskriver alle parametrene, der er overført fra formularen i afsnit 3.2.4), skrevet i JSTL:

```
<!-- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%> -->
<html>
<head><title>Parametre3 med JSTL</title></head>
<body>

<p>
Her er parameteren "navn": <c:out value="${param['navn']}"/><br>
Parameteren på en anden måde: <c:out value="${param.navn}"/><br>
</p>

<p>
Andre parametre:<br>
<c:forEach var="parametren" items="${param}">
  <c:out value="${parametren.key}"/>
  med værdi: '<c:out value="${parametren.value}"/>'.
 <br>
</c:forEach>
</p>

<p>
Parameteren 'kan_spise' har flere værdier.<br>
De er:
<c:forEach var="vaerdi" items="${paramValues['kan_spise']}">
  <c:out value="${vaerdi}"/>
</c:forEach>
</p>

</body>
</html>
```

Læg mærke til, at vi her bruger af <c:out /> for at skrive ud:

```
Parameteren på en anden måde: <c:out value="${param.navn}"/><br>
```

Ved at bruge <c:out /> bliver specielle tegn som <, >, &, ' og " erstattet af deres HTML−entiteter (d.v.s. <, >, & o.s.v.).

Vi *kunne* godt bruge et EL−udtryk direkte i HTML−koden, som vi plejer:

```
Parameteren på en anden måde: ${param.navn}<br>
```

men da ville indholdet af navn−parameteren kunne fortolkes som HTML−kode med de problemer, det medfører (se afsnit 8.5.1, HTML−injektion).

Det er derfor en rigtig god idé, at udnytte JSTLs indbyggede sikkerhedsfunktioner og altid bruge <c:out />, hver gang man udskriver data, der kommer fra brugeren.

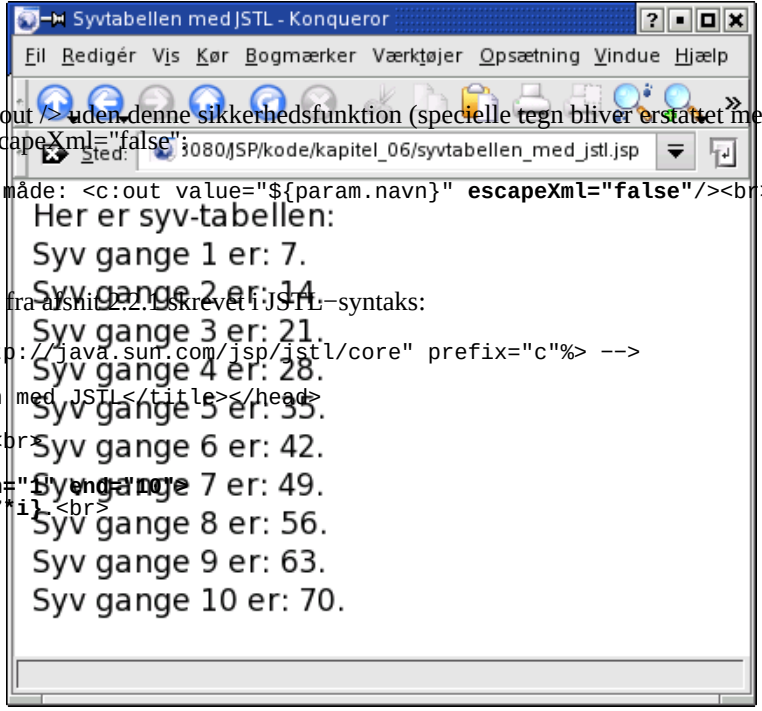
Hvis man ønsker at bruge <c:out /> uden denne sikkerhedsfunktion (specielle tegn bliver erstattet med deres HTML−entiteter) kan det slås fra med attributten escapeXml="false":

```
Parameteren på en anden måde: <c:out value="${param.navn}" escapeXml="false"/><br>
```

6.1.5 Løkker

Her er eksemplet Syvtabellen fra afsnit 2.2. Skrevet i JSTL−syntaks:

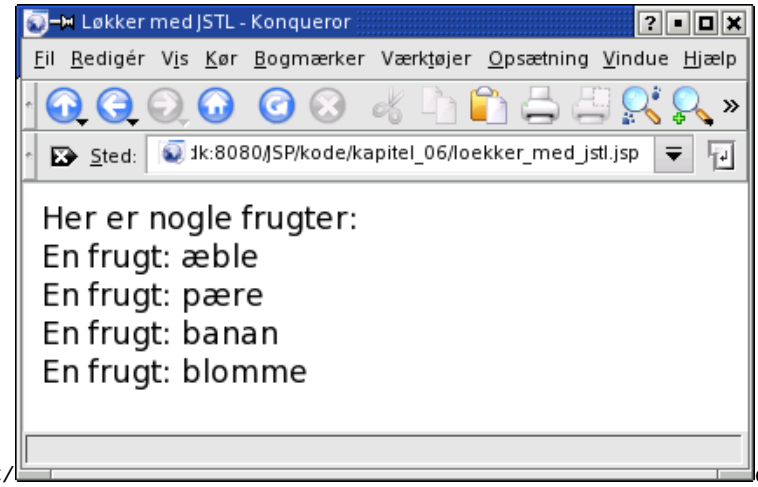
```
<!-- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%> -->
<html>
<head><title>Syvtabellen med JSTL</title></head>
<body>
<p>Her er syv−tabellen:<br>

<c:forEach var="i" begin="1" end="10">
  Syv gange ${i} er: ${7*i} <br>
</c:forEach>

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



Man kan også gennemløbe andet end tal. I afsnit 6.1.4 findes et eksempel hvor alle parametrene gennemløbes.

Herunder løber vi gennem nogle strenge med navne på frugter:

```
<!-- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%> -->
<html>
<head><title>Løkker med JSTL</title></head>
<body>
<p>Her er nogle frugter:<br>

  <c:forEach var="f" items="æble,pære,banan,blomme">
    En frugt: ${f}<br>
```




```
</c:forEach>
</p>
</body>
</html>
```

6.1.6 Oversigt

Her er en oversigt over alle koderne i dette core−tagbiblioteket.

<c:set

Sætter/opretter en variabel til et udtryk. Virkefelt kan angives med scope=... der kan være page, request, session eller application.

<c:remove

Fjerner en variabel (fra et bestemt virkefelt hvis scope=... er angivet).

<c:out

Udskriver et udtryk/en variabel. Det svarer altså til <%= ... >, blot erstattes alle specielle tegn med HTML−entiteter (kan slås fra med escapeXml="false")

<c:import url=...

Henter indholdet af en absolut eller relativ URL og putter ind i siden eller i en variabel hvis var=... og scope=... er angivet.

<c:param name=...

(inde i <c:import>)

Tilføjer parameter til en <c:import>−forespørgsel.

<c:redirect url=...

Omdirigerer til en anden URL.

<c:url

Definerer en URL.

<c:if test=...

Betingelse. Kroppen udføres, hvis betingelsen er opfyldt.

<c:choose

Vælger mellem et antal alternativer, mærket med <when> og <otherwise>

<c:when test=...

(inde i <c:choose>)

Kroppen udføres, hvis betingelsen er opfyldt.

<c:otherwise

(inde i <c:choose>)

Kroppen udføres, hvis ingen af betingelserne i <c:when var opfyldt.

<c:forEach

Gennemløber noget.

<c:forTokens items=... delims=...

Opdeler en streng (items=) i bidder efter nogle skilletegn (delims=) og gennemløber bidderne.

<c:catch

Fanger alle undtagelser, der sker mellem <c:catch> og </c:catch>.
Med <c:catch var=...> kan undtagelsen gemmes i en variabel.

6.2 Internationalisering og formatering (<fmt: >)

JSTL har en lang række funktioner til at parse og formatere tal og datoer. De ligger i taglib−bet fmt, som importeres med:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
```

6.2.1 Oversigt

Her er en oversigt over alle koderne i dette tagbibliotek.

<fmt:requestEncoding value=...

Sætter tegnsætter for anmodningen.

<fmt:setLocale value=...

Sætter sproget sådan, at formateringer sker efter dette sprog (f.eks.: value="da_DK")

<fmt:timeZone value=...

Sætter tidszonen for alle operationen i kroppen

<fmt:setTimeZone value=...

Sætter tidszonen globalt

<fmt:bundle basename=...

Indlæser et resursebundt, der kan bruges i kroppen.

<fmt:setBundle basename=...

Indlæser et resursebundt globalt.

<fmt:message key=...

Slår op i resursebundt og formaterer og udskriver resultatet

<fmt:param>

(inde i <fmt:message>)

Supplerer et opslag med <fmt:message> med et argument

<fmt:formatNumber

Formaterer et tal

<fmt:parseNumber

Fortolker en streng som et tal

<fmt:formatDate value=...

Formaterer en dato/tid

<fmt:parseDate

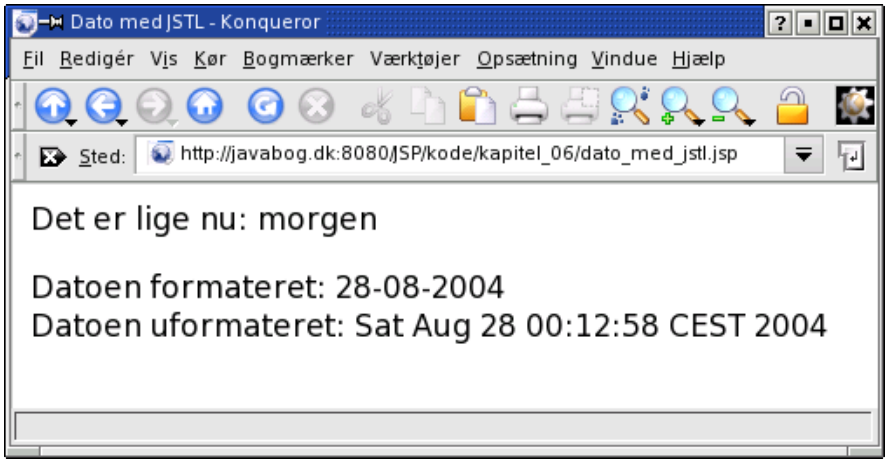
Fortolker en streng som en dato/tid

6.2.2 Eksempel på brug

Her er et eksempel på, hvordan man arbejder med datoer:

```
<!-- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
     <%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%> -->
<html>
<head><title>Dato med JSTL</title></head>
<body>
<p>
  Det er lige nu:
  <jsp:useBean id="tid" class="java.util.Date" />
  <c:set var="t" value="${tid.hours}" />
  <c:choose>
    <c:when test="${t <= 9}">morgen</c:when>
    <c:when test="${t <= 12}">formiddag</c:when>
    <c:when test="${t <= 17}">eftermiddag</c:when>
    <c:when test="${t <= 21}">aften</c:when>
    <c:otherwise>nat</c:otherwise>
  </c:choose>
</p>

<p>
  Datoen formateret: <fmt:formatDate value="${tid}" /><br>
  Datoen uformateret: ${tid}<br>
</p>
</body>
</html>
```

Sammenlign med eksemplet i afsnit 2.3.



Først opretter vi et Date−objekt og knytter det til variablen 'tid':

```
  <jsp:useBean id="tid" class="java.util.Date" />
```

I almindelig JSP ville man skrive 'new Date()'. Den specielle måde med <jsp:useBean /> diskuteres i afsnit 6.8, Kommunikation mellem JSTL og Java):

Så aflæser vi, hvad klokken er med ${tid.hours} (svarer til 'tid.getHours()' i almindelig JSP) og sætter resultatet ind i variablen 't':

```
  <c:set var="t" value="${tid.hours}" />
```

Derefter vælges mellem et sæt alternativer (svarer til en switch−sætning eller en række kædede if−sætninger i almindelig JSP):

```
  <c:choose>
    <c:when test="${t <= 9}">morgen</c:when>
```

Til sidst udskriver vi tiden formateret med

```
<fmt:formatDate value="${tid}" />
```

6.3 JSTL og databaser (<sql: >)

JSTL har nogle meget effektive kommandoer til databasekommunikation. De importeres med:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
```

For at få adgang til en database fra JSTL, kan databasedriveren og −URLen angives i siden sammen med JSP−koden:

```
<sql:setDataSource url="jdbc:mysql:///test" driver="com.mysql.jdbc.Driver" />
```

Det er dog langt bedre, at angive det i web.xml under intialiseringsparameteren med navn 'javax.servlet.jsp.jstl.sql.dataSource':

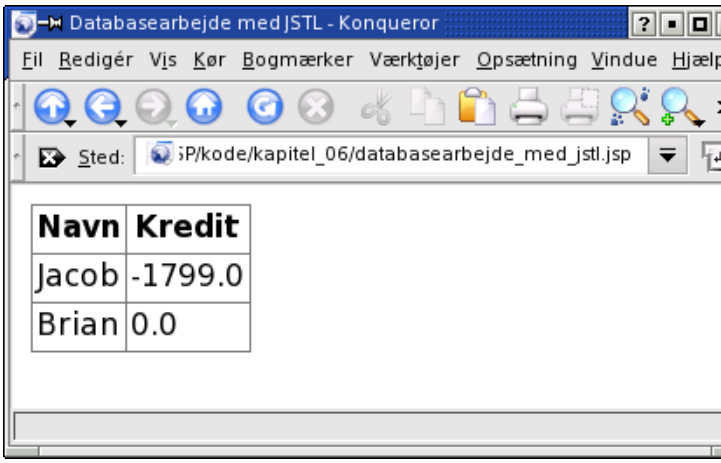
```
  <context-param>
    <param-name>javax.servlet.jsp.jstl.sql.dataSource</param-name>
    <param-value>jdbc:mysql:///test,com.mysql.jdbc.Driver</param-value>
  </context-param>
```

Herunder har vi angivet "jdbc:mysql:///test" og "com.mysql.jdbc.Driver". Hvis man ville angive brugernavn og adgangskode, f.eks. 'j' og 'hemli' skulle parameteren have været:

```
    <param-value>jdbc:mysql:///test,com.mysql.jdbc.Driver,j,hemli</param-value>
```

Her ses et eksempel på brug:

```
<%-- Anvend JSTLs core- og database-tagbibliotek --%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

<html>
<head><title>Databasearbejde med JSTL</title></head>
<body>

<%-- datakilde angives i web.xml, men kan også direkte på siden (kun til test):
<sql:setDataSource url="jdbc:mysql:///test" driver="com.mysql.jdbc.Driver" /> --%>

<%-- lav en forespørgsel --%>
<sql:query var="kunder" sql="SELECT * FROM kunder" />

<table border="1" cellpadding="2" cellspacing="0">
  <tr>
    <th>Navn</th>
    <th>Kredit</th>
  </tr>

  <%-- Gå gennem alle kunderne --%>
```




```
  <c:forEach var="kunde" begin="0" items="${kunder.rows}">
    <tr>
      <td>${kunde.navn}</td>
      <td>${kunde.kredit}</td>
    </tr>
  </c:forEach>
</table>

</body>
</html>
```

6.3.1 Oversigt

Her er en oversigt over alle koderne i sql-tagbiblioteket.

<sql:transaction

Udfører alt kode i kroppen som én databasetransaktion

<sql:query

Udfører SQL-forespørgslen i kroppen eller i attributten sql=...

<sql:update

Udfører SQL-sætningen i kroppen eller i attributten sql=...

<sql:param

Sætter en parameter i en SQL-sætning.

<sql:dateParam

Sætter en parameter i en SQL−sætning (for Date−objekter).

<sql:setDataSource

Sætter database−URLen (kun til udvikling, ikke egnet til drift)

6.4 XML−behandling (<x: >)

JSTL har fin understøttelse for behandling af XML−data. De har præfikset 'x' og importeres med:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x"%>
```

6.4.1 Syndikering med XML−funktionerne i JSTL

Dette afsnit forudsætter, at du har læst afsnit 11.3, Syndikering (nyhedsfødning).

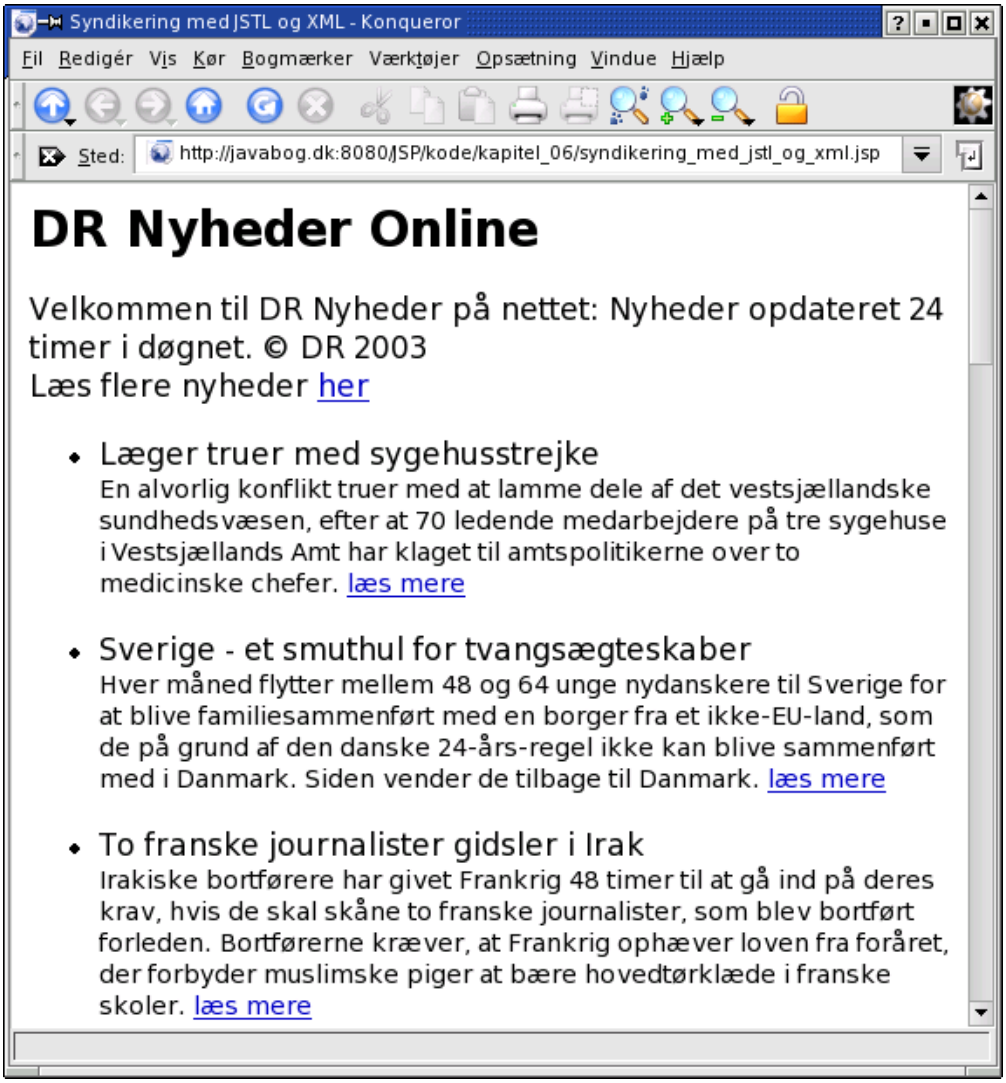
Indholdssyndikering (fødning af nyheder fra andre websider ind i ens egne) er beskrevet i afsnit 11.3. Man kan udnytte JSTLs XML−funktioner til at flette andres RSS−filer med nyheder ind i ens egne sider.

Herunder et eksempel, hvor vi fletter vi nyheder fra Danmarks Radio ind.

Først importeres core− og XML−taglibbet. Derefter hentes en ekstern webside med nyhederne med <c:import>, der efter parses som XML med <x:parse> og lægges i variablen rss.

Denne variabel bruges derefter til, at navigere rundt i DOM−træet og med passende XPath−udtryk gennemløbe elementer med <x:forEach> og udskrive elementer med <x:out>.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x"%>
<html>
<head><title>Syndikering med JSTL og XML</title></head>
<body>

<c:import var="rssKilde" url="http://www.dr.dk/nyheder/html/nyheder/rss/"/>
<x:parse var="rss" doc="${rssKilde}"/>

<h1>
  <x:out select="$rss//*[name()='channel']/*[name()='title'][1]"/>
</h1>
<x:out select="$rss//*[name()='channel']/*[name()='description'][1]"/>
<br>
Læs flere nyheder
<a href="<x:out select="$rss//*[name()='channel']/*[name()='link'][1]"/>">her</a>

<ul>
  <x:forEach select="$rss//*[name()='item']">
    <li>
      <x:out select="./*[name()='title']"/><br>
      <font size="−1">
        <x:out select="./*[name()='description']" escapeXml="false" />
        <a href="<x:out select="./*[name()='link']"/>">læs mere</a><br><br>
```




```
    </font>
    </li>
  </x:forEach>
</ul>

</body>
</html>
```

6.4.2 Caching af nyhedskilder

Kommandoerne

```
<c:import var="rssKilde" url="http://www.dr.dk/nyheder/html/nyheder/rss/"/>
<x:parse var="rss" doc="${rssKilde}"/>
```

henter nyhederne fra DR og fortolker XML−koden, hver gang der spørges på siden.

Ved mange forespørgsler kan dette føre til lange svartider og unødig stor belastning af nyhedskilden. Det kunne derfor være en idé, at cache det fortolkede XML−dokument i f.eks. en time.

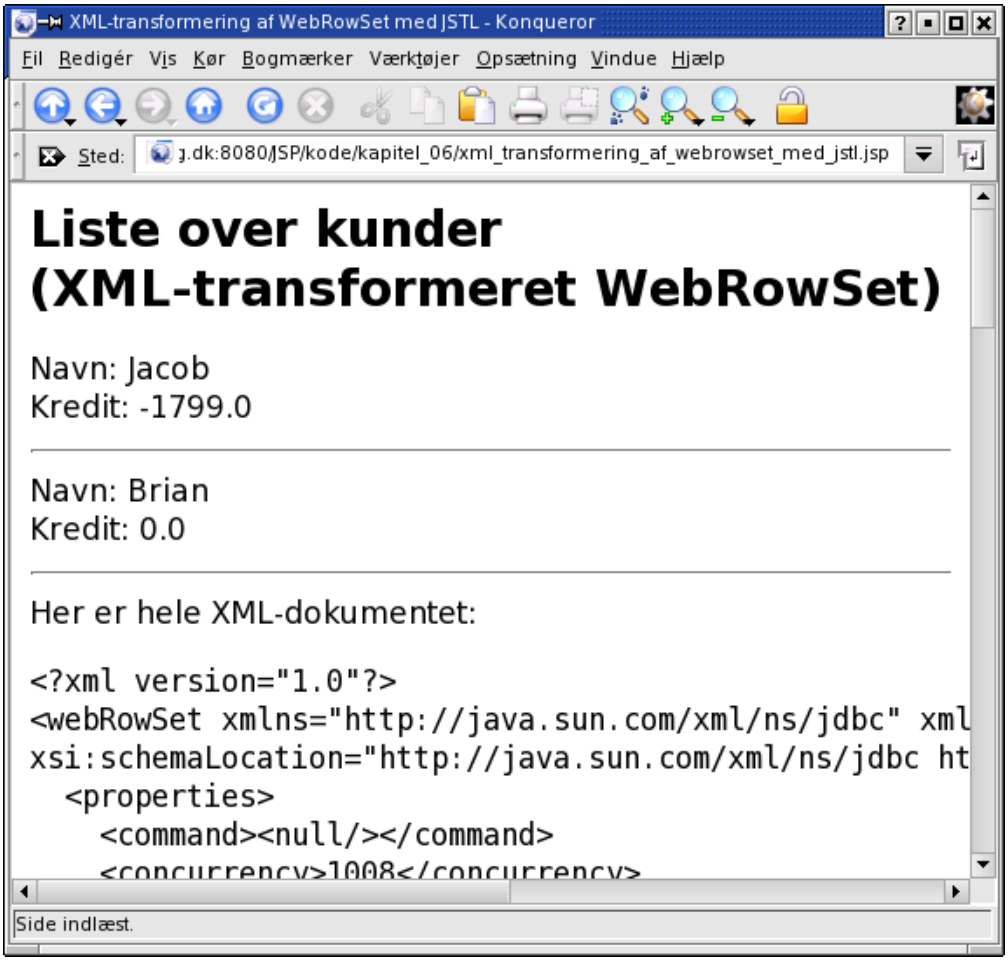
I JSTL kan det gøres ved at gemme det fortolkede dokument i application−objektet sammen med en tidsstempel. Ved en anmodning, kontrolleres det, om der er gået mere end en time (3600000 millisekunder) og kun da hentes XML−dokumentet fra serveren:

```
<jsp:useBean id="nu" class="java.util.Date"/>
<c:if test="${nu.time > cachetid-3600000}">
  <c:import var="rssKilde" url="http://www.dr.dk/nyheder/html/nyheder/rss/"/>
  <x:parse var="rss" doc="${rssKilde}" scope="application"/>
  <c:set var="cachetid" value="${nu.time}" scope="application"/>
</c:if>
```

6.4.3 WebRowSet og XML−transformering med JSTL

Lad os nu se på, hvordan man kunne transformere et XML−dokument om til HTML.

I afsnit 5.9.4 så vi, hvordan et WebRowSet−objekt kunne repræsenterer et sæt rækker fra en database som XML. I det følgende eksempel indlæses og behandles dette XML−dokument, for at producere en HTML−side med kunderne:



Her hentes først data i et WebRowSet, der skrives ned i variablen kundeXml, der er oprettet med <jsp:useBean>, så den både er tilgængelig for almindelig Javakode og for JSTL (j.v.f. afsnit 6.8, Kommunikation mellem JSTL og Java). Derefter parses den med <x:parse> og gennemløbes og alle rækkerne udskrives.

```
<%@ page language="java" import="java.sql.*, javax.sql.*, com.sun.rowset.*" %>
<%-- Bemærk: rowset.jar fra Sun skal være i CLASSPATH --%>

<%-- Anvend JSTLs core- og XML-tagbibliotek --%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x"%>

<html>
<head><title>XML-transformering af WebRowSet med JSTL</title></head>
<body>
<%
  Class.forName("com.mysql.jdbc.Driver");
  Connection con = DriverManager.getConnection("jdbc:mysql:///test");
  Statement stmt = con.createStatement();
  ResultSet rs = stmt.executeQuery("SELECT * FROM kunder");

  // Oprettelse fejler nogen gange første gang (uvist hvorfor)
  try { new WebRowSetImpl(); } catch (Exception e) { } // ignorér fejlen
  WebRowSetImpl wrs = new WebRowSetImpl();

  //anden mulig løsning, se  http://forum.java.sun.com/thread.jspa?threadID=540624&tstart=270 (xxx ikke afprøvet
  //java.util.Locale loc =  Locale.getDefault();
  //java.util.Locale.setDefault(Locale.US);
  //WebRowSetImpl wrs = new WebRowSetImpl();
  //java.util.Locale.setDefault(loc);

  wrs.populate(rs);
  rs.close();
  // wrs.writeXml(System.out); // Skriv evt XML i webserverens logfil
%>

<%-- Skriv XML-streng til variablen kundeXml --%>
<jsp:useBean id="kundeXml" class="java.io.StringWriter" />
<% wrs.writeXml(kundeXml); %>

<%-- Fortolk XML-streng og put det parsede DOM-træ ind i kunder.
     Vi får data ud at kundeXml (der er af typen StringWriter) til en streng
     ved at sætte den sammen med streng " " (ét mellemrum): "${kundeXml} " --%>
<x:parse var="kunder" doc="${kundeXml} "/>

<h1>Liste over kunder (XML-transformeret WebRowSet)</h1>

<%-- Løb gennem alle rækkerne i tabellen og udskriv 1. og 2. kolonne --%>
```

```
<x:forEach select="$kunder//*[name()='currentRow']">
  Navn: <x:out select="*[1]"/><br>
  Kredit: <x:out select="*[2]"/><br>
  <hr>
</x:forEach>

Her er hele XML-dokumentet:
<pre>
<c:out value="${kundeXml}" />
</pre>

</body>
</html>
```

6.4.4 Oversigt

Her er en oversigt over alle koderne i XML-tagbiblioteket:

<x:parse

Parser et XML-dokument, der kommer i kroppen eller som attributten 'source'.

<x:set

Sætter en variabel til resultatet af et XPath-udtryk

<x:out

Udskriver noget, ligesom <%= ... >, men evaluerer XPath-udtryk.

<x:forEach

Gennemløber et XPath-udtryk.

<x:if test=...

Betingelse. Kroppen udføres, hvis XPath-udtrykket evaluerer til 'true'

<x:choose

Vælger mellem et antal alternativer, mærket med <when> og <otherwise>

<x:when test=...
(inde i <x:choose>)

Kroppen udføres, hvis betingelsen er opfyldt.

<x:otherwise
(i <x:choose>)

Kroppen udføres, hvis ingen af betingelserne i <c:when> var opfyldt.

<x:transform

Transformerer et XML-dokument med et XSLT stylesheet

<x:param
(i <x:transform>)

Tilføjer parameter til en <x:transform>-operation

6.5 Forskellige funktioner (<fn: >)

En række forskellige funktioner til at arbejde med strenge, kan importeres med:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
```

Derefter kan man skrive fn:funktionsnavn(...) inde i et EL-udtryk, for at kalde funktionen.

F.eks. kunne man i forrige eksempel skrive kundenavnet med store bogstaver med:

```
      <td>${fn:toUpperCase(kunde.navn)}</td>
```

Man kunne i afsnit 6.1.4 udskrive længden af parameteren navn med:

```
Længden af parameteren navn: ${fn:length(param.navn)}
```

eller måske kontrollere længden af det indtastede navn:

```
<c:if test="${fn:length(param.navn) < 2}" >
  Du skal indtaste et navn på mindst to tegn
</c:if>
```

6.5.1 Oversigt

Her er en oversigt over funktionerne:

length(objekt)	Finder længden af objektet. Er objektet en streng, giver det antallet af tegn. Er objektet et array eller en anden datastruktur (collection), giver det antallet af elementer.
contains(streng, delstreng) containsIgnoreCase(streng, delstreng) startsWith(streng, delstreng) endsWith(streng, delstreng)	Undersøger om en streng indeholder en delstreng.
escapeXml(streng)	Erstatter specielle tegn med deres HTML−entitet sådan, at tegn i strengen ikke kan (mis)fortolkes som HTML−kode.
indexOf(streng, delstreng)	Finder indekset (placeringen) i en streng af en delstreng.
join(streng[], mellemstreng)	Sætter et array af strenge sammen til én streng.
replace(streng, førstr, efterstr)	Giver en streng, hvor alle forekomster af delstrengen førstr er erstattet med strengen efterstr.
split(streng, separatorstreng)	Opdeler en streng i et array af delstrenge.
substring(streng, startpos, slutpos)	Giver delstrengen fra startpos til og med slutpos.
substringBefore(streng, delstreng) substringAfter(streng, delstreng)	Giver den del af strengen der er før (hhv. efter) delstrengen.
toUpperCase(streng) toLowerCase(streng)	Giver en streng hvor alle tegn er lavet om til store (hhv. små) bogstaver.
trim(streng)	Fjerner blanktegn fra begge ender af strengen.

6.6 Installation af JSTL og EL

For at webserveren kender taglibbet, skal JAR−filerne jstl.jar og standard.jar være tilgængelig for webapplikationen, f.eks. ved at de ligger i WEB−INF/lib/.

Du kan f.eks. kopiere dem fra webapps/jsp−examples/WEB−INF/lib/ i JSP−eksemplerne, der fulgte med i Tomcat.

Hvis du har problemer, så se i afsnit 4.9.6, hvordan du kan lægge JAR−filerne ind.

6.6.1 Versionsproblemer med JSTL og EL

JSTL og EL har udviklet sig lidt knudret: Fra at være bygget sammen til at være adskilte ting, der kan bruges uafhængigt af hinanden. Nu (med JSP 2.0) er EL bygget direkte ind i webserveren, mens JSTL er en JAR−fil, der skal være tilgængelig for webapplikationen. Derfor kan man komme ud for en del bøvl med JSTL og EL i nogle kombinationer.

Man skal her være opmærksom på, at ens web.xml−fil kan være afgørende for, om tingene virker som forventet, idet web.xml i version 2.3 som udgangspunkt *ikke* understøtter EL, mens version 2.4 gør.

Har man en web.xml i version 2.3, kan det være at man er nødt til at skrive EL−udtryk ud med JSTL−koden <c:out />:

```
Det svarer til <c:out value="${12*alder}" /> måneder.<br>
```

Det bedste er at opgradere sin web.xml−fil. Her ses starten af en web.xml−fil i version 2.3 (der får webserveren til kun at fortolke EL−udtryk, der er inde i JSTL−koder):

web.xml i version 2.3

```
<?xml version = '1.0' encoding = 'ISO−8859−15'?>
<!DOCTYPE web−app PUBLIC "−//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web−a
<web−app>
```

Disse linjer skal erstattes med det følgende (web.xml i version 2.4 − du kan kopiere det fra den web.xml, der følger med eksemplerne i Tomcat):

web.xml i version 2.4

```
<?xml version="1.0" encoding="ISO−8859−1"?>
<web−app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
```

```
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">
```

Endelig skal man være opmærksom på, at JSTL-taglib-bibliotekerne nu ligger under 'jsp':

```
<%-- Anvend JSTL 1.1's core- og XML-tagbibliotek (JSP 2.0) --%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x"%>
```

hvor man før i tiden skrev f.eks.:

```
<%-- Anvend JSTL 1.0's core- og XML-tagbibliotek (JSP 1.1)  --%>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jstl/xml" prefix="x"%>
```

6.7 JSTL versus almindelig javakode i JSP

Når nu man kan det samme med almindelig javakode i JSP, hvorfor så lære et nyt sprog og begynde at programmere i JSTL i stedet?

- JSTL er simplere end Java. Man kan ikke nær så mange ting i JSTL som i almindelig javakode og syntaksen er simplere.
- For ikke-Java-kyndige har JSTL en betydelig lettere syntaks end hvis de skulle lave almindelig javakode i JSP.
- JSTL er en anelse langsommere end almindelig javakode.
- JSTL har mange funktioner til at hente data ud fra java-objekter og er velegnet til at lave en adskillelse af præsentation (HTML-kode) og forretningslogik (der i større projekter nemmest skrives som selvstændige java-klasser uden for JSP-siderne).
- Hvis man laver et større projekt, der involverer at HTML-designere også skal redigere i dokumenterne, er JSTL velegnet, da JSTL overholder XML-syntaksen. De fleste HTML-editorer lader derfor JSTL-koderne være (og tillader endda at redigere i dem), mens almindelig javakode mellem <% og %> ofte går tabt[2].

Det anbefales, at bruge JSTL til *præsentation af informationer*. Det er næppe en god idé at skrive indviklet forretningslogik i JSTL, der er Java mere velegnet.

6.8 Kommunikation mellem JSTL og Java

Bemærk at JSTL og almindelig javakode i JSP fungerer på forskellige måder og at det sjældent er nogen god idé, at have begge slags kode i samme JSP-sider.

Man kan for eksempel ikke uden videre bruge en variabel fra JSTL i Java eller omvendt.

Således vil JSP-koden

```
<%
    Date tid = new Date();
  %>
```

der opretter et Date-objekt og knytter det til variablen 'tid', *ikke* gøre den tilgængelig i JSTL.

Det skal man efterfølgende gøre ved, at gemme den som attribut til page, request, session eller application-objektet. Følgende vil således fungere fint:

```
<%
    Date tid = new Date();
    page.setAttribute("tid", tid );  // lav variablen 'tid' i JSTL
  %>
  Klokken er ${tid}
```

6.8.1 Javabønner

Man kan også bruge javabønne-syntaksen (javabønner forklares i kapitel 9):

```
  <jsp:useBean id="tid" class="java.util.Date" />
```

Man kan eventuelt give javabønnen et virkefelt, der giver mulighed for at genbruge det samme objekt næste gang siden besøges (virkefelter er beskrevet senere, i afsnit 9.2.5, Oversigt over de mulige virkefelter).

Her gives f.eks. sessionen som virkefelt:

```
  <jsp:useBean id="tid" class="java.util.Date" scope="session" />
```

6.8.2 Implicit definerede objekter i EL

Når man skriver et udtryk i EL, kan man vælge præcist, hvad man vil have fat i, ved at benytte de følgende implicitte variabler:

- pageScope, der giver adgang til attributter gemt i page-objektet
- requestScope, der giver adgang til attributter gemt i request-objektet
- sessionScope, der giver adgang til attributter gemt i session-objektet
- applicationScope, der giver adgang til attributter gemt i application-objektet

- pageContext, der søger alle ovenstående objekter igennem for attributter
- param, der giver adgang til parametre i request−objektet
- paramValues, der giver adgang til parameterværdierne i request−objektet
- header, der giver adgang til HTTP−headere i anmodningen
- headerValues, der giver adgang til værdierne af HTTP−headere i anmodningen
- cookie, der giver adgang til cookier, der fulgte med anmodningen
- initParam, der giver initialieringsparametre for webapplikationen (i web.xml)

Starter et EL−udtryk ikke med et af ovenstående ord, vil EL søge igennem alle implicitte variabler og derefter søge page−, request−, session− og application−objektets attributter igennem efter et objekt med det pågældende navn.

6.9 Mere læsning

- Du kan læse mere om JSTL på:
  http://java.sun.com/products/jsp/jstl/
- Om JSTL i 'The J2EE Tutorial':
  http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JSTL.html
- Oversigten over alle JSTL−koderne kan ses på:
  http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/

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

2Desværre med én undtagelse: Importeringen af et taglib foretages med koden <%@ taglib ... %>, der ikke overholder XML−syntaksen og ikke overlever i en HTML−designer. Det kan man dog omgå, ved at lægge den ind i en HTML−kommentar (<!−− ... −−>), sådan her:
<!−− <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%> **−−>**

7 Inde i webserveren

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

Det forudsætter kapitel 3, Interaktive sider, og kapitel 5, Brug af databaser.

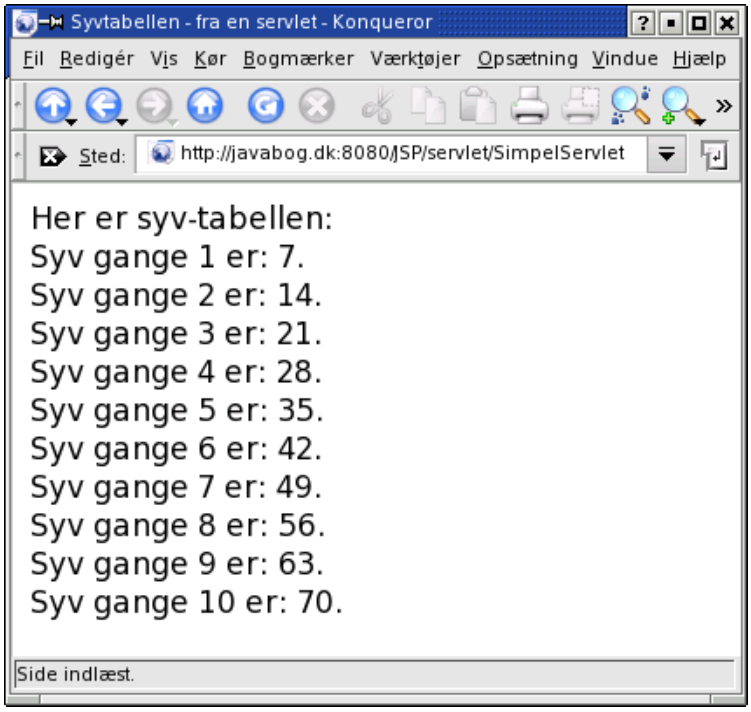
Dette kapitel handler om konfiguration af webserveren og nogle af de ting, der sker "under motorhjelmen", inde i webserveren.

7.1 Servletter

En servlet er en Java−klasse, der bliver brugt af en webserver. Den har noget programkode, der opbygger et HTML−dokument, som bliver sendt til klienten.

Her er en servlet, der udskriver det samme som JSP−siden syvtabellen.jsp (fra afsnit 2.2.1, Indlejrede java−udtryk):

```
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 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>");
  }
}
```



Læg mærke til:

- Servlet−klassen skal arve fra HttpServlet
- Servletten skal have en doGet(request, response)−metode[1]
- Ud fra response−objektet får man et out−objekt, som man kan bruge til at skrive HTML−kode til klienten/netlæseren.

Uddata fra en servlet er som regel HTML, men kan egentlig være af enhver slags. Servletten kan sende andre slags data, e.v.t. binære, eller viderestille til en anden adresse.

Når webserveren får en anmodning om en URL, der svarer til servlettens navn[2], kalder den metoden doGet() på servletten (allerførste gang opretter serveren et objekt af den pågældende klasse). Denne metode får overført alle relevante oplysninger om anmodningen i et request−objekt og skal så udfylde et response−objekt med svaret til klienten.

For at afprøve servletten (forudsat vi har oversat til binær kode og installeret det i en webserver) kalder vi f.eks. adressen http://localhost:8080/JSP/servlet/SimpelServlet op i en netlæser (man ser at servlettens navn normalt indgår i URLen).

7.1.1 Anmodningsmetoder

Som diskuteret i afsnit 3.6.4, Skjule parametrene (POST−metoden), findes der flere metoder en netlæser kan tage i anvendelse, når den skal foretage en forespørgsel (GET og POST).

GET−anmodninger forårsager at servlettens doGet(request, response)−metode bliver kaldt.

Tilsvarende forårsager POST−anmodninger, at metoden doPost(request, response) kaldes.

Ofte er man dog ligeglad med, om anmodninger kommer med den ene eller anden metode og så lader man blot doPost() kalde doGet(), som så gør arbejdet:

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response) throws IOException
  {
    doGet(request, response);
  }
```

Hvilken anmodningsmetode, der bruges, kan altid ses med request.getMethod().

7.1.2 Hvornår bruge JSP og hvornår bruge servletter

Servletter er som sagt javakode (Java−klasser), der genererer HTML−kode.

JSP er på en måde det omvendte: HTML−kode med indlejret Javakode.

Er du i tvivl om du skal bruge JSP−sider eller servletter til en given opgave, så er her et par betragtninger, du kan tage med i dine overvejelser:

- Som enhver anden Java−klasse skal en servlets .java−fil oversættes til en binær .class−fil, som skal lægges i WEB−INF/classes/
- Alle servletter skal nævnes i WEB−INF/web.xml (se afsnit 7.2, Installation af en servlet)
- JSP−sider kan genkendes og redigeres visuelt i de fleste ordentlige HTML−redigeringsværktøjer. Det vil sige, at man ikke behøver at skrive al HTML−koden selv.

- Servletter kan, som alle andre klasser, genkendes og redigeres af Java−udviklingsværktøjer. Værktøjerne kan dog ikke hjælpe dig med at lave HTML−koden i servletten.

Alt i alt er servletter mest velegnede til indviklet programlogik, mens JSP−sider er velegnede, hvis der skal produceres meget HTML til klienten. Er du begynder, bør du nok vælge JSP−sider, da de er lettest at komme i gang med.

7.2 Installation af en servlet

Før du kan køre en servlet, skal du angive servlet−klassen, der svarer til en URL på webserveren. Det gøres i filen web.xml (der skal ligge i mappen WEB−INF), hvor der skal stå:

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>

Her er et uddrag af web.xml. Gennem navnet "En simpel servlet" bindes klassen SimpelServlet til URLen /servlet/SimpelServlet:

```
<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>
```

7.2.1 Flere URLer til samme servlet

Denne syntaks løsriver klassenavnet fra URLen og giver dermed mulighed for mange fikse måder at sætte sin webapplikation op på.

Vi kunne f.eks. sende alle forespørgsler til /simpel/* til vores SimpelServlet ved at tilføje:

```
  <servlet-mapping>
    <servlet-name>En simpel servlet</servlet-name>
    <url-pattern>/simpel/*</url-pattern>
  </servlet-mapping>
```

Herefter ville f.eks. http://localhost:8080/JSP/simpel/hvad.som.helst få webserveren til at kalde SimpelServlet.

Et praktisk eksempel på, hvor dette kunne være nyttigt, findes i afsnit 10.5.3, Eksempel på en Frontkontrol.

7.2.2 Avanceret: Automatisk binding af servletter

7.3 Avanceret: JSP−siders interne virkemåde

7.3.1 Kigge i de genererede servletter

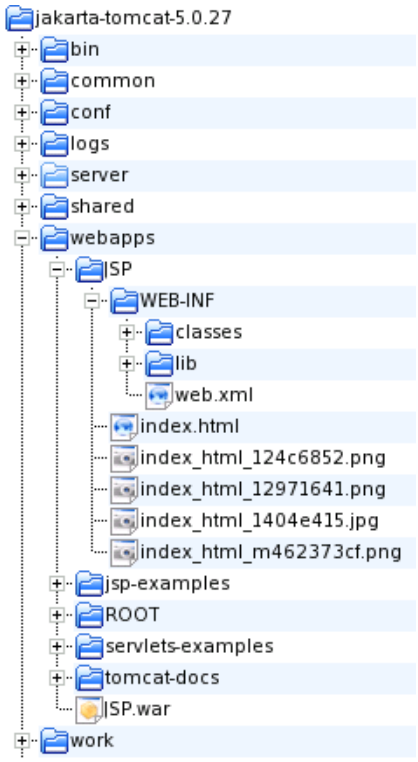
7.3.2 Eksempel

7.3.3 JSP−siders livscyklus

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

7.4 Webapplikationer

En *webapplikation* er en gruppe websider, der hører sammen. En webserver kan godt have flere webapplikationer kørende samtidig og oprette/installere, fjerne og opdatere webapplikationer uafhængigt af hinanden.



På harddisken har serveren normalt hver webapplikation liggende i en separat mappe. I Tomcat ligger disse mapper under webapps/.

På billedet til højre kan man på mappestrukturen se, at der er installeret 5 webapplikationer, nemlig "JSP", "jsp−examples", "ROOT", "servlet−examples" og "tomcat−docs".

Disse kan findes under en URL, der svarer til navnet. F.eks. ligger webapplikationen "jsp−examples" på http://localhost:8080/jsp−examples/ (eneste undtagelse er ROOT som ligger i roden, på http://localhost:8080).

HTML− og JSP−sider, billeder etc. ligger som filer i webapplikationens mappe. I den specielle undermappe WEB−INF/ ligger alle andre filer:

I **WEB−INF/classes** skal dine egne klasser ligge.

I **WEB−INF/lib** skal du lægge JAR−filer med funktioner, der bruges i dine JSP−sider.

I **WEB−INF/web.xml** skal driftsbeskrivelsen til serveren ligge (se afsnit 7.5).

Nogen gange kan filstrukturen afvige fra, hvad man forventer ud fra URLen, fordi det står angivet i WEB−INF/web.xml (se afsnit 7.2). Det gælder især servletter. Således kunne SimpelServlet, der ligger webapplikationen "JSP" i filen WEB−INF/classes/SimpelServlet.class, være afbildet over på URL−adressen http://localhost:8080/JSP/servlet/SimpelServlet .

7.4.1 Pakkede webapplikationer (WAR−filer)

En webapplikation kan være pakket ned i en WAR−fil (Web ARchive), der er en ZIP−fil med en mappestruktur, der svarer til, hvordan filerne skal ligge på serveren.

Installere en WAR−fil

En WAR−fil kan kopieres ind i mappen webapps i Tomcat. Derefter vil webserveren selv pakke den ud og idriftsætte den.

På billedet ovenfor kan man nederst se hvordan WAR−filen JSP.war er blevet kopieret til webapps/JSP.war, hvorefter webserveren har pakket ud i webapps/JSP/ , så den er tilgængelig på adressen http://localhost:8080/JSP/.

Lave en WAR−fil

Selvom princippet i WAR−filer egentlig er enkelt nok, er der mange, der til at starte med, har problemer med at lave WAR−filer. Bruger du et udviklingsværktøj, bør du se, om det kan hjælpe dig med at generere WAR−filen. Ellers er det nemmeste ofte, at tage

en eksisterende WAR−fil (tjek at den virker!) og erstatte filerne i den med dine egne.

Filstrukturen og mappestrukturen i en WAR−fil skal være præcist som den på harddisken, især skal man huske at WEB−INF/ *skal* ligge i roden af ZIP−filen. Her er et eksempel:

```
index.html
index_html_124c6852.png
index_html_12971641.png
index_html_1404e415.jpg
WEB-INF/
 WEB-INF/classes/
 WEB-INF/classes/javabog/Login.class
 WEB-INF/classes/kalender/Bruger.class
 WEB-INF/classes/kalender/Kalender.class
 WEB-INF/classes/minPakke/Person.class
 WEB-INF/classes/SimpelServlet.class
 WEB-INF/lib/
 WEB-INF/lib/activation.jar
 WEB-INF/lib/mail.jar
 WEB-INF/lib/mysql-connector-java-3.0.9-stable-bin.jar
 WEB-INF/web.xml
```

7.5 Samlet driftsbeskrivelse (web.xml)

Det er i webapplikationens WEB−INF/web.xml man beskriver, hvordan serveren skal køre ens webapplikation i drift.

Her ses hvordan driftsbeskrivelsen web.xml samlet ser ud for eksemplerne til denne bog.

web.xml − driftsbeskrivelse for webapplikationen

```
<?xml version="1.0" encoding="ISO−8859−1"?>
<web−app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web−app_2_4.xsd"
    version="2.4">

  <!−− Samlet eksempel på konfigurationsfil for en webapplikation.
       Filnavn: /WEB−INF/web.xml
    −−>

  <display−name>Eksempler fra javabog.dk</display−name>
  <description>Alle eksemplerne fra bogen "Webprogrammering med Java Server Pages",
    der også kan læses på http://javabog.dk/JSP</description>

  <!−− ====================================================================
       Eksempler på initialiseringsparametre − se afsnit 4.5.5 i bogen
       Aflæs med f.eks: application.getInitParameter("dbDriver")
       ==================================================================== −−>
  <context−param>
    <description>Driveren til databasen</description>
    <param−name>dbDriver</param−name>
    <param−value>com.mysql.jdbc.Driver</param−value>
  </context−param>

  <context−param>
    <description>Adressen (URLen) på databasen</description>
    <param−name>dbUrl</param−name>
    <param−value>jdbc:mysql:///test</param−value>
  </context−param>

  <context−param>
    <description>Brugernavnet til databasen</description>
    <param−name>dbBruger</param−name>
    <param−value>root</param−value>
  </context−param>

  <context−param>
    <description>Adgangskoden til databasen</description>
    <param−name>dbAdgangskode</param−name>
    <param−value></param−value>
  </context−param>

  <!−− Denne initialiseringsparameter er til JSTLs databasetags i afsnit 6.3 −−>
  <context−param>
    <param−name>javax.servlet.jsp.jstl.sql.dataSource</param−name>
    <param−value>jdbc:mysql:///test,com.mysql.jdbc.Driver</param−value>
  </context−param>

  <!−− Disse initialiseringsparametre er til Login−bønnen i afsnit 9.5 −−>
  <context−param>
    <description>SMTP−server for epost fra Login−bønnen i afsnit 9.5</description>
    <param−name>postserver</param−name>
    <param−value>post.tele.dk</param−value>  <!−− TDC som internetudbyder −−>
  </context−param>
```

```
<context-param>
  <description>Afsenderadressen for Login-bønnen i afsnit 9.5</description>
  <param-name>postafsender</param-name>
  <param-value>din@adresse.dk</param-value>
</context-param>

<!-- ====================================================================
     Indstillinger til afsnit 7.1, Servletter
     ==================================================================== -->
<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>

<!-- Alle forespørgsler til /simpel/* sendes til vores SimpelServlet -->
<servlet-mapping>
  <servlet-name>En simpel servlet</servlet-name>
  <url-pattern>/simpel/*</url-pattern>
</servlet-mapping>

<!-- Bind alle klasser (herunder også servletter) til /servlet/Klassenavn -->
<!-- Kommenteret bort af sikkerhedshensyn
<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>
-->


<!-- ====================================================================
     Indstillinger til afsnit 8.2, Adgangskontrol
     ==================================================================== -->
<!-- De roller, der er aktuelle for webapplikationen -->
<security-role>
  <role-name>kunde</role-name>
  <role-name>administrator</role-name>
</security-role>

<!-- En gruppe sider med adgangbegrænsning -->
<security-constraint>
  <display-name>Sikkerhedsbegraensning i kapitel 8</display-name>

  <!-- Præcist hvilke sider, der er omfattet af adgangbegrænsningen -->
  <web-resource-collection>
    <url-pattern>/kode/kapitel_08/beskyttet_side.jsp</url-pattern>
  </web-resource-collection>

  <!-- Hvilke brugerroller har adgang til de pågældende sider -->
  <auth-constraint>
    <role-name>kunde</role-name>
  <role-name>administrator</role-name>
  </auth-constraint>

  <!-- Hvordan skal data sendes over netværket -->
  <!-- Mulighederne er: NONE, INTEGRAL og CONFIDENTIAL (kræver SSL)
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
  -->
</security-constraint>

<!-- Brugeren skal identificeres med en login-formular -->
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/kode/kapitel_08/loginside.jsp</form-login-page>
    <form-error-page>/kode/kapitel_08/fejlagtig_login.jsp</form-error-page>
  </form-login-config>
  <!-- Andre muligheder er (giver et loginvindue i stedet for login-formular)
  <auth-method>BASIC</auth-method>
  <auth-method>DIGEST</auth-method>
  <auth-method>CLIENT-CERT</auth-method>
  -->
</login-config>


<!-- ====================================================================
     Indstillinger til kapitel 10, Arkitekturer i webprogrammering
     ==================================================================== -->

<!-- Forespørgsler til /kode/kapitel_10/ sendes til kontrol.jsp -->
<servlet>
```

```
    <servlet-name>Kontrolloer</servlet-name>
    <jsp-file>/kode/kapitel_10/kontrol.jsp</jsp-file>
  </servlet>

  <servlet-mapping>
    <servlet-name>Kontrolloer</servlet-name>
    <url-pattern>/kode/kapitel_10/</url-pattern>
  </servlet-mapping>

  <!-- Frontkontrol: Alt der starter med /bank sendes til kontrol.jsp -->
  <servlet>
    <servlet-name>Frontkontrol</servlet-name>
    <jsp-file>/WEB-INF/bank/kontrol.jsp</jsp-file>
  </servlet>

  <servlet-mapping>
    <servlet-name>Frontkontrol</servlet-name>
    <url-pattern>/bank/*</url-pattern>         <!-- Bemærk: * i URL-mønster -->
  </servlet-mapping>

</web-app>
```

7.6 Test dig selv

7.7 Resumé

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

1Man kan også definere andre metoder, se senere.

2I afsnit 7.2 vil vi se, hvordan man binder en servlet til en URL.

8 Sikkerhed og adgangskontrol

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.1 Kryptering og SSL

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.1.1 I gang med HTTPS

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.1.2 HTTPS−standardporten

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.2 Adgangskontrol

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.2.1 Containerstyret adgangskontrol

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.2.2 Hurtigt i gang med adgangskontrol

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.3 Datakilder til adgangskontrol

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.3.1 Brugerdata fra tekstfil

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.3.2 Andre datakilder

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.3.3 Brugerdata fra database

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.4 Former for brugeridentifikation

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.4.1 Kryptering af brugerdata (transportgarantier)

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.4.2 Eksempel: Formularbaseret login

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.5 Sikkerhed i en webapplikation

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.5.1 HTML−injektion

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.5.2 SQL−injektion

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.5.3 Cross−site scripting (XSS) og sessionskapring

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.5.4 Eksempel på sessionskapring

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.5.5 Måder at beskytte sig mod sessionskapring

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.5.6 Bortfiltrering af ulovlige tegn

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.5.7 Opgave: Lave en sikker gæstebog

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

8.5.8 Løsning: Sikker gæstebog

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

9 Javabønner i JSP−sider

Dette kapitel forudsættes i resten af bogen. Det forudsætter kapitel 4, Videre med JSP, og kapitel 5, Brug af databaser.

Det er ønskværdigt at adskille tekstligt indhold og programkode fra hinanden, sådan at f.eks. en HTML−designer kan koncentrere sig om JSP−sidernes HTML−layout og indhold, mens en programmør koncentrerer sig om den bagvedliggende kode.

Et af de vigtigste elementer i sådan en struktur er, at programkode og datastrukturer lægges for sig i separate javaklasser, som så kan bruges fra JSP−siderne. Disse javaklasser anvendes derefter som *javabønner* fra JSP−siderne.

9.1 Javabønner

Enhver java−klasse kan kaldes en javabønne (eng.: Java Bean), bare to krav er opfyldt:

- Klassen er defineret i en pakke.
- Klassen har en konstruktør uden parametre.

De fleste klasser − både systemets og dem du selv definerer − kan altså opfattes som javabønner.

9.1.1 Bruge javabønner fra en JSP−side

Den mest almindelige måde at bruge en javabønne på er, at knytte den til brugerens session, sådan at den følger med brugeren. Det gøres med JSP−koden:

```
<jsp:useBean id="bønnenavn" class="pakkenavn.Klassenavn" scope="session" />
```

Eksempelvis kunne vi knytte en ArrayList (eller Vector) til brugerens session med:

```
<jsp:useBean id="liste" class="java.util.ArrayList" scope="session" />
```

Derefter kan ArrayList−objektet bruges under navnet 'liste'.

Attributten scope="session" betyder, at objektet har sessionen som virkefelt, d.v.s. at objektet vil blive husket og genbrugt næste gang brugeren besøger siden og at hver bruger vil have sit eget objekt (som beskrevet i afsnit 4.1, Sessioner).

> <jsp:useBean /> bruges ikke kun til at oprette et objekt, det bruges også til efterfølgende at genbruge objektet

Man kan også knytte objekter til andet end sessionen (beskrives senere, i afsnit 9.2.5).

9.1.2 Ønskeseddel med ArrayList som javabønne

Her ses ønskeseddel−eksemplet fra afsnit 4.1.1 igen, blot ændret til at bruge klassen ArrayList som en javabønne, der er knyttet til brugerens session (ændringerne er i fed).

```
<html>
<head><title>Ønskeseddel − ArrayList som javabønne</title></head>
<body>
Dette eksempel demonstrerer, hvordan et ArrayList−objektet kan knyttes til
brugerens session som en javabønne.

<jsp:useBean id="ønsker" class="java.util.ArrayList" scope="session" />

<h3>Skriv et ønske</h3>
Skriv noget, du ønsker.
<form>
<input type="text" name="oenske">
</form>
<%
  // se om der kommer en parameter med endnu et ønske
  String ønske = request.getParameter("oenske");
  if (ønske != null) {
    ønsker.add(ønske);                      // tilføj ønske til listen
  }

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

Sammenlignet med det oprindelige eksempel i afsnit 4.1.1 ser man, at al arbejdet med at hente listen ud af brugerens session (og evt. oprette listen) er sparet væk.

9.2 At definere en javabønne og bruge den

Lad os nu se på, hvordan man definerer sine egne javabønner. Vi definerer klassen Person:

Person.java

```
package minPakke; //oversat fil skal ligge i WEB−INF/classes/minPakke/Person.class

public class Person
{
  private String fornavnet;           // intern variabel (ikke egenskab)
  private int alderen;                // intern variabel (ikke egenskab)

  public Person()                     // konstruktør uden parametre skal eksistere
  {
  }

  public String getFornavn()          // aflæser egenskaben fornavn
  {
```

```
    return fornavnet;
  }

  public void setFornavn(String n)    // sætter egenskaben fornavn
  {
    fornavnet = n;
  }

  public int getAlder()               // aflæser egenskaben alder
  {
    return alderen;
  }

  public void setAlder(int n)         // sætter egenskaben alder
  {
    alderen =  n;
    System.out.println("Alder sat til: "+n); // udskriv ændring til loggen
  }
}
```

9.2.1 Pakker og filplaceringer for klasser

Vil man bruge en klasse fra en JSP−side, skal klassen defineres i en pakke[1]. Derfor er bønnen defineret i pakken minPakke.

Webserveren leder kun efter javaklasser i undermappen WEB−INF/classes/. Klasserne skal derfor ligge dette specielle sted, for at kunne bruges fra JSP−siderne.

Således skal den oversatte klasse, Person.class, ligge i WEB−INF/classes/minPakke/ i den webapplikation klassen tilhører.

9.2.2 Egenskaber på en javabønne

Ofte har en javabønne nogle get− og set−metoder, som kan ændre i objektets data. Sådanne metoder kaldes også *egenskaber* (eng.: properties).

Herunder har vi for eksempel bønnen Person med metoderne getFornavn() og setFornavn(), svarende til egenskaben *fornavn* og de tilsvarende metoder for egenskaben *alder*.

Læg mærke til, at bønnen har egenskaben *alder* af typen int, fordi den har metoderne

```
  public int getAlder()
  public void setAlder(int n)
```

Hvordan data gemmes internt er underordnet (i dette tilfælde gemmes det i en privat variabel kaldet alderen − data kunne lige så godt være lagret på en anden måde, f.eks. i form af fødselsåret).

Tilsvarende har bønnen egenskaben *fornavn* af typen String, fordi den har metoderne

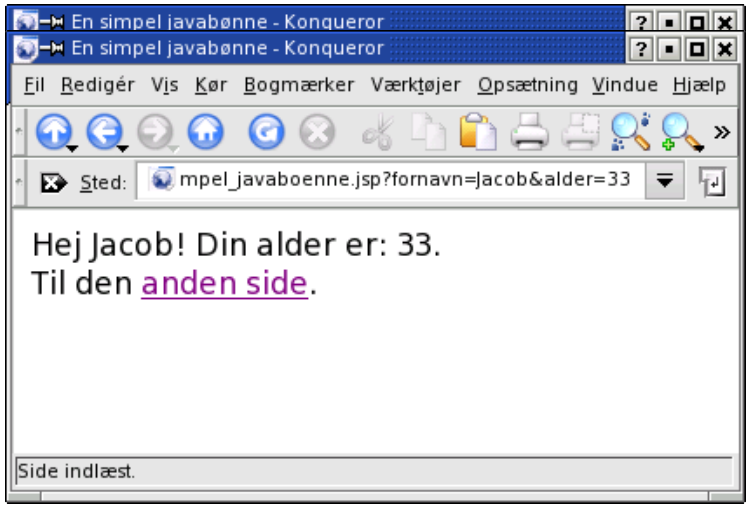
```
  public String getFornavn()
  public void setFornavn(String n)
```

Serveren kan automatisk kalde egenskabernes set−metoder, hvis der kommer data fra en formular, hvor parameternavnene passer med bønnens egenskaber.

9.2.3 Sætte egenskaber fra en JSP−side

En JSP−side, der bruger bønnen, kunne have en formular med parametrene *fornavn* og *alder* overført. Den kunne se således ud:

```
<html>
<head><title>En simpel javabønne</title></head>
<body>
<jsp:useBean id="person" class="minPakke.Person" scope="session" />
<jsp:setProperty name="person" property="*" />

<% if (person.getFornavn() == null) { %>
  Indtast dit fornavn og din alder:
  <form>
    <input type="text" name="fornavn">
    <input type="text" name="alder" size="4">
    <input type="submit" value="OK">
  </form>
<% } else { %>
  Hej <%= person.getFornavn() %>!
  Din alder er: <%= person.getAlder() %>.
<% } %>
<br>Til den <a href="en_simpel_javaboenne_2.jsp">anden side</a>.
</body>
</html>
```



Udfylder man formularen, ser man, at dens indhold kommer over i javabønnen (i serverens log kan man også se at metoden setAlder() bliver kaldt).

Det skyldes linjen, der aflæser parametrene (fornavn og alder) og kalder set−metoder i bønnen (setFornavn() og setAlder()):

```
<jsp:setProperty name="person" property="*" />
```

Faldgruber

Når du definerer egenskaber og vil binde dem til formularfelter, så husk:

> Formularfelter bør altid skrives med *lille* startbogstav. De tilsvarende metoder i bønnen bør altid skrives med *stort* startbogstav efter get/set

Forklaringen er, at formularfeltet (som bliver navnet på parameteren og egenskaben) har stort startbogstav i metodenavnet efter get/set. Hedder egenskaben f.eks. 'fornavn' kommer metoderne i bønnen til at hedde getFornavn() og setFornavn().

Brug derfor små bogstaver i formularfeltet, f.eks :

```
<input type="text" name="fornavn">  <%-- rigtigt --%>
```

Brug aldrig f.eks.

```
<input type="text" name="Fornavn">  <%-- forkert! --%>
```

og heller ikke:

```
<input type="text" name="FORNAVN">  <%-- forkert! --%>
```

En anden hyppig fejltagelse er at glemme formen af metoderne beskrevet i afsnit 9.2.2.

9.2.4 Aflæse bønnens egenskaber

Egenskaber kan også aflæses for bønnen og indsættes i HTML−koden. Med f.eks.:

```
Din alder er: <jsp:getProperty name="person" property="alder" />.
```

bliver værdien af bønnens egenskab *alder* sat ind.

Det er kortere simpelt hen at skrive f.eks.:

```
Din alder er: <%= person.getAlder() %>.
```

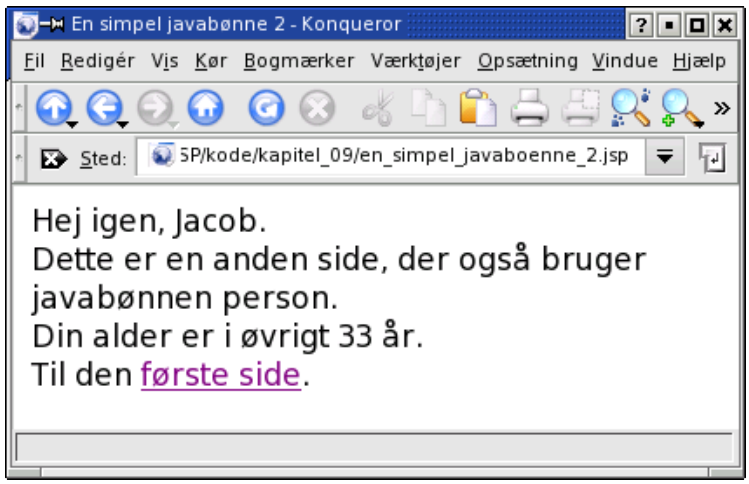
Derfor ses koden <jsp:getProperty ... /> i praksis ikke så ofte.

9.2.5 Virkefelter for javabønner

Her er en anden side, der også bruger Person−bønnen.

```
<html>
<head><title>En simpel javabønne 2</title></head>
<body>
<jsp:useBean id="person" class="minPakke.Person" scope="session" />

Hej igen, <%= person.getFornavn() %>.<br>
Dette er en anden side, der også bruger javabønnen person.<br>
Din alder er i øvrigt <%= person.getAlder() %> år.<br>
Til den <a href="en_simpel_javaboenne.jsp">første side</a>.
</body>
</html>
```

Besøger man denne, vil man se, at dataene fra en_simpel_javaboenne.jsp er gemt:



Det skyldes, at bønnen har sessionen som virkefelt (eng.: scope). Objektet vil blive husket i brugerens session og genbrugt, når brugeren besøger andre sider, der har koden

```
<jsp:useBean id="person" class="minPakke.Person" scope="session" />
```

Oversigt over de mulige virkefelter

Virkefeltet scope="session", der knytter bønnen til session−objektet (se afsnit 4.5.4, session − objekt der følger den enkelte bruger) er det mest anvendte, men der findes også andre virkefelter.

Eksempelvis vil scope="application", gemme bønnen i application−objektet (se afsnit 4.5.5, application − fælles for hele webapplikationen) og dermed give bønnen hele applikationen som virkefelt, dvs. at den samme bønne deles mellem alle brugere og alle sider (analogt til en global variabel).

Derudover kan en javabønne have virkefeltet scope="request". Da knyttes bønnen til den aktuelle anmodning (til request−objektet, se afsnit 4.5.1, request − anmodningen fra klienten) og bliver smidt væk, når anmodningen er fuldført. Det kan bruges ved server−omdirigering fra en side til en anden (se afsnit 4.3.2, Server−omdirigering).

Det mest kortlivede virkefelt er scope="page", der knytter bønnen til page−objektet. Bønnen eksisterer indtil udførelsen af den aktuelle side er fuldført (omdirigeres der til andre sider smides bønnen væk). Den har altså samme levetid som en almindelig variabel erklæret i JSP−siden imellem <% og %>.

9.2.6 Avanceret: Virkemåden af <jsp:useBean... />

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

9.2.7 Initialisering af javabønner

Ofte ønsker man at få noget programkode kørt én gang når javabønnen initialiseres. Det kan man gøre ved at lægge koden mellem <jsp:useBean> og </jsp:useBean>. Herunder skriver vi ud hver gang et person−objekt bliver oprettet og sætter fornavnet til "(ukendt)":

```
  <jsp:useBean id="person" class="minPakke.Person" scope="session" >
    <%
      out.print("Et nyt person−objekt blev oprettet!");
      person.setFornavn("(ukendt)");
```

```
    %>
  </jsp:useBean>
```

Et eksempel på dette er vist i afsnit 10.4.4.

9.2.8 Avanceret: Virkemåden af <jsp:setProperty .../>

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

9.3 Egenskaber på en javabønne

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

9.3.1 Indekserede egenskaber

Derudover kan egenskaber være *indekseret*. En egenskab siges at være indekseret, hvis HTML−sidens formular har flere parametre, der hedder det samme.

Herunder er et eksempel på brug af indekserede egenskaber.

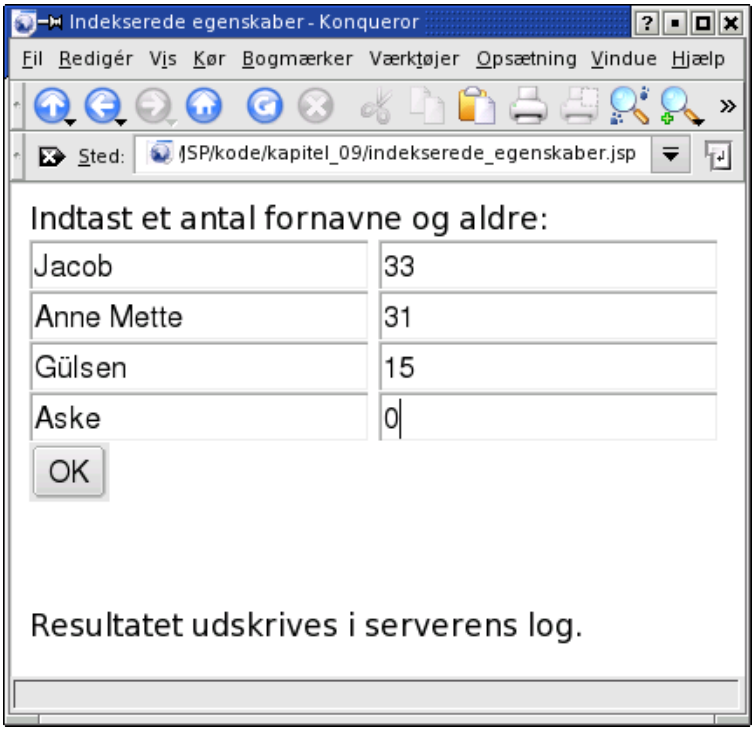
```
<html>
<head><title>Indekserede egenskaber</title></head>
<body>
<jsp:useBean id="ie" class="minPakke.IndekseredeEgenskaber" scope="session" />
<jsp:setProperty name="ie" property="*" />
Indtast et antal fornavne og aldre:
<form>
  <input type="text" name="fornavn"> <input type="text" name="alder"><br>
  <input type="text" name="fornavn"> <input type="text" name="alder"><br>
  <input type="text" name="fornavn"> <input type="text" name="alder"><br>
  <input type="text" name="fornavn"> <input type="text" name="alder"><br>
  <input type="submit" value="OK">
</form>
<br><br>
Resultatet udskrives i serverens log.
</body>
</html>
```

Javabønnen har defineret de indekserede egenskaber *fornavn* og *alder* til at modtage parametrene:

IndekseredeEgenskaber.java

```
package minPakke;
public class IndekseredeEgenskaber
{
  public void setFornavn(String[] arr)  // sætter indekseret egenskab fornavn
  {
    for (int i = 0; i < arr.length; i++) {
      System.out.println("setFornavn(arr["+i+"] = "+arr[i]+")");
    }
  }

  public void setAlder(int[] arr)       // sætter indekseret egenskab alder
  {
    for (int i = 0; i < arr.length; i++) {
      System.out.println("setAlder(arr["+i+"] = "+arr[i]+")");
    }
  }
}
```

Lad os nu forestille os at brugeren udfylder formularen med f.eks.:



Da vil javabønnen udskrive i serverens log (for Tomcat hedder den logs/catalina.out):

```
setAlder(arr[0] = 33)
setAlder(arr[1] = 31)
setAlder(arr[2] = 15)
setAlder(arr[3] = 0)
setFornavn(arr[0] = Jacob)
setFornavn(arr[1] = Anne Mette)
setFornavn(arr[2] = Gülsen)
setFornavn(arr[3] = Aske)
```

9.4 Eksempel: En dagskalender

Dette eksempel er et mindre program, der viser en dagskalender og giver mulighed for at redigere i den.

9.4.1 Udseende

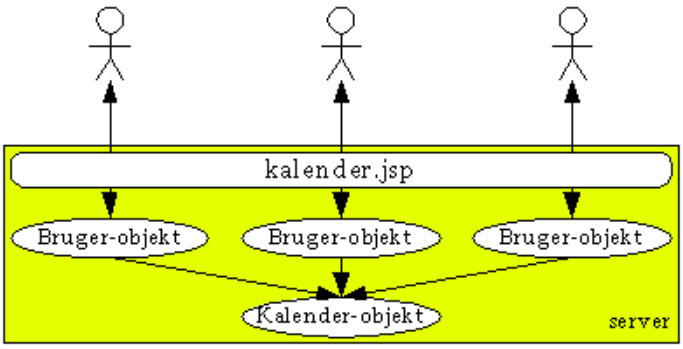
Som udgangspunkt ser kalenderen således ud:

Klikker man på "rediger" kommer samme skærmbillede frem, men denne gang med indtastningsfelter så man kan rette i teksten:

9.4.2 Programmets virkemåde

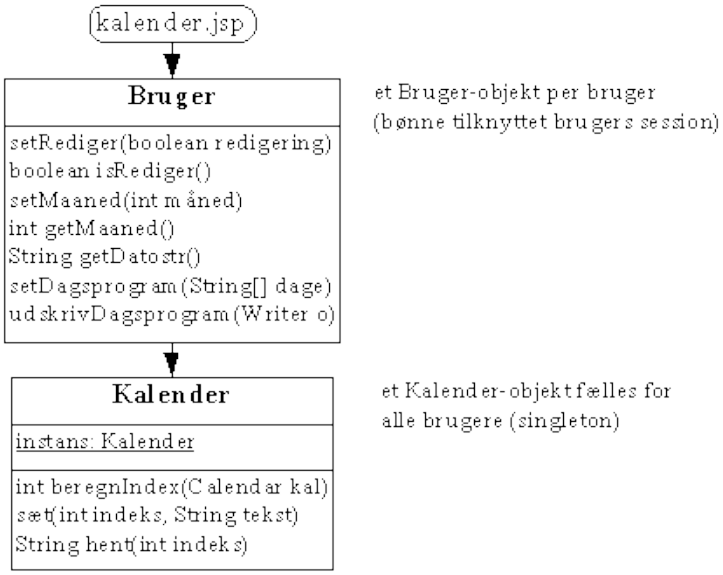
Programmet består af JSP−siden kalender.jsp og klasserne Bruger.java og Kalender.java.

Klassen Bruger er en javabønne, der knyttes til brugerens session. Hver bruger får derfor sit eget Bruger−objekt. Alle Bruger−objekterne kommunikerer med det samme Kalender−objekt (derved bliver det en fælles kalender).



Kalender−objektet repræsenterer logikken i programmet og er ikke specielt rettet mod en webapplikation, men kunne også bruges i andre sammenhænge.

Her er klassediagrammet (private metoder og variabler er ikke vist):



9.4.3 HTML−siden (kalender.jsp)

Først JSP−siden som brugeren ser:

kalender.jsp

```
<%-- Knyt bønnen Bruger til brugerens session under navnet b --%>
<jsp:useBean id="b" class="kalender.Bruger" scope="session" />

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

<html>
<head><title>Kalender</title></head>
<body>

<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); %>
```

```
<%
  }
%>
</body>
</html>
```

Bemærk hvordan vi definerer variablen b af type Bruger og opretter et objekt, der genbruges i hele den kommende session:

```
<jsp:useBean id="b" class="kalender.Bruger" scope="session" />
```

JSP−siden benytter Bruger−bønnens egenskaber *rediger*, *maaned* og *dagsprogram*. Vi sørger for at egenskaber på b bliver sat, hvis de tilsvarende parametre overføres til siden med:

```
<jsp:setProperty name="b" property="*" />
```

der (j.v.f. afsnit 9.2.6) svarer til koden:

```
  String parm1 = request.getParameter("rediger");
  if (parm1 != null) b.setRediger( parm1.equals("true") );

  String parm2 = request.getParameter("maaned");
  if (parm2 != null) b.setMaaned( Integer.parseInt(parm2) );

  String[] parm3 = request.getParameterValues("dagsprogram");
  if (parm3 != null) b.setDagsprogram  ( parm3 );
```

9.4.4 Bruger−bønnen (Bruger.java)

Herunder er Bruger−bønnens. Metoder der svarer til egenskaber er fremhævet med fed:

Bruger.java

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

public class Bruger {
  private SimpleDateFormat månedFormat;
  private SimpleDateFormat dagugedagFormat;

  public void setLocale(Locale sproget) {
    månedFormat = new SimpleDateFormat("MMMM yyyy",sproget); // f.x. 'maj 2004'
    dagugedagFormat = new SimpleDateFormat("dd EE",sproget); // f.x. '31 ma'
  }

  // tom konstruktør − ellers er det ikke en javabønne
  public Bruger() { setLocale(new Locale("da","DK")); }

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

  private GregorianCalendar dato = new GregorianCalendar();

  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.sætDagstekst(start+i,dagsprogram[i]);
  }

  /** Producerer HTML−kode der viser 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>\n<code>");
      out.write(dagugedag);
      out.write("</code> ");
      if (!redigering) out.write(Kalender.instans.hentDagstekst(i+start));
      else out.write("<input type='text' size=30 name='dagsprogram' value='"
        + Kalender.instans.hentDagstekst(i+start).replace('\'','\"') + "'>");
      kal.roll(Calendar.DAY_OF_MONTH,1);
```

```
    }
  }
}
```

Bemærk hvordan egenskaben *dagsprogram* svarer til metoden setDagsprogram() og inputfeltet i HTML−formularen med navnet dagsprogram. Egenskaben er et eksempel på en *indekseret* egenskab, da den sættes med et array (se afsnit 9.3.1, Indekserede egenskaber).

Selve HTML−koden til dagsprogrammet bliver ikke produceret af JSP−siden. I stedet kaldes metoden udskrivDagsprogram() med out−objektet som parameter på Bruger−objektet, der så producerer HTML−koden.

Man kunne også havde ladet JSP−siden producere HTML−koden til kalenderen (det ville nok være bedre stil, men ville også gøre eksemplet sværere at overskue).

9.4.5 Kalender−objektet (Kalender.java)

Den anden klasse er et Kalender−objekt, der fungerer som data−lager og husker aftalerne.

Kalender.java

```
package kalender;

import java.util.*;
import java.text.*;
import java.beans.*;
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−2003)*366+dag;
  }

  /**
   * Sæt teksten for en bestemt dag.
   * @see #beregnIndeks(Calendar)
   * @param indeks Indekset i listen. Skal først findes med beregnIndeks()
   * @param tekst Teksten for dagen
   */
  public void sætDagstekst(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 hentDagstekst(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 XML−fil på disken (hvis den findes)
      XMLDecoder kal = new XMLDecoder(new FileInputStream("kalender.xml"));
      // alternativ: hent serialiseret objekt i stedet for XML−data
      //ObjectInputStream kal = new ObjectInputStream(
      //                        new FileInputStream("kalender.ser"));
      liste = (ArrayList) kal.readObject();
      kal.close();
      System.out.println("Kalender indlæst: "+liste);
    } catch (Exception e) {
      System.out.println("Kalender ikke indlæst, opretter ny: "+e);
      liste = new ArrayList();
    }
    GemRegelmæssigt g = new GemRegelmæssigt(); // gemmer kalenderen på disken
  }

  /** Sørger for at gemme kalenderen regelmæssigt (i en separat tråd) */
  class GemRegelmæssigt extends Thread
  {
    public GemRegelmæssigt() {
```

```
      setDaemon(true);   // systemet må godt stoppe selvom tråden stadig kører
      setPriority(MIN_PRIORITY);
      start();
    }

    public void run() {
      while (true) try {
        Thread.sleep(1*60*1000); // hvert minut,
        if (ændret) {            // hvis kalenderen er ændret
          ændret = false;        // gem kalenderen på disken
          // gem som XML
          XMLEncoder kal = new XMLEncoder(new FileOutputStream("kalender.xml"));
          // alternativ: gem som serialiseret objekt i stedet for XML
          //ObjectOutputStream kal = new ObjectOutputStream(
          //                         new FileOutputStream("kalender.ser"));
          kal.writeObject(liste);
          kal.close();
          System.out.println("Kalender gemt: "+liste);
        }
      } catch (Exception e) { e.printStackTrace(); }
    }
  }
}
```

Denne kalender er fælles for alle brugerne og er programmeret sådan, at der kun bliver oprettet ét objekt af typen Kalender (en singleton).

I stedet for at bruge en database henter og gemmer kalenderen en gang imellem sine data i en XML−fil (med navnet 'kalender.xml'). Formatet og indholdet af XML−filen diskuteres i afsnit 11.2.2, Nem generering af XML fra Java−objekter. Kommenteret væk er også kode til i stedet at gemme data serialiseret i en fil (med navnet 'kalender.ser').

9.5 Eksempel: Login og brugeroprettelse

Det følgende eksempel viser, hvordan man kan have beskyttede sider, hvor brugerne skal logge ind med brugernavn og adgangskode, defineret i en database.

Derudover kan nye brugere registrere sig, ved at opgive en epost−adresse, hvorefter en bruger vil blive oprettet i databasen og en adgangskode vil blive sendt til epost−adressen.

Hvis brugeren ønsker det, kan hans oplysninger gemmes i en cookie, sådan at han automatisk kan blive logget ind, næste gang han besøger siden.

9.5.1 Brug af eksemplet

Eksemplet er udformet således, at du kan tage det og bruge det i dit eget projekt.

Bemærk dog, at der i næsten alle webservere allerede findes foruddefinerede måder at lave beskyttede sider på, blot ved at ændre i konfigurationsfilerne, som beskrevet i afsnit 8.2.1, Containerstyret adgangskontrol.

Eksemplet her er derfor reelt kun nødvendigt at kopiere, hvis du ønsker f.eks. dynamisk brugeroprettelse med tilsending af adgangskoder eller du selv vil styre det præcise udseende af login−skærmbilledet eller en af de andre ting nævnt i afsnit 8.2, Adgangskontrol.

9.5.2 Login−bønnen

Først javabønnen Login.java. Den har egenskaberne *brugernavn og adgangskode*, der skal være korrekt sat før egenskaben *ok* bliver sand og brugeren får lov til at komme videre.

Metoderne, der svarer til egenskaber, er fremhævet med fed.

Login.java

```
package javabog;

import java.sql.*;
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.servlet.*;
/**
 * Denne javabønne har ansvaret for registrering og logintjek af en bruger.
 * Husk at mail.jar, activation.jar og MySQL−driveren skal være i CLASSPATH.
 */
public class Login
{
  private String brugernavn = "";
  private String adgangskode = "";
  private String epost = "";
  private String meddelelse = "";       // fejlmeddelelse til brugeren

  private boolean tjek = false;         // om adgangskode skal tjekkes
  private boolean loggetInd = false;    // om adgangskoden var korrekt
```

```
public void setBrugernavn(String bn)  { tjek=true; brugernavn=bn; }
public String getBrugernavn()         { return brugernavn; }

public void setAdgangskode(String ak) { tjek=true; adgangskode = ak; }

public void setEpost(String epost)    { this.epost = epost; }
public String getEpost()              { return epost; }

public void setMeddelelse(String m)   { meddelelse = m; }
public String getMeddelelse() { String m=meddelelse; meddelelse=""; return m; }

/** Egenskaben loggetInd. Kan af sikkerhedsgrunde kun aflæses */
public boolean isLoggetInd() {
  if (tjek) return false;       // er der sket ændringer skal der logges ind
  return loggetInd;
}

/** Forbindelsen til databasen. Oprettes når klassen indlæses */
private static Connection con = null;
private static String postserver = null;
private static String postafsender = null;

/** Initialisering. Skal kaldes før bønnen bruges */
public synchronized static void init(ServletContext application)
{
  if (con!=null) return; // initialisér kun hvis det ikke allerede er gjort

   // Opret forbindelse til databasen når klassen bruges første gang
  try {
    String drv=null, url=null, bru=null, adg=null;
    if (application!=null) {
      drv = application.getInitParameter("dbDriver");
      url = application.getInitParameter("dbUrl");
      bru = application.getInitParameter("dbBruger");
      adg = application.getInitParameter("dbAdgangskode");
      postserver = application.getInitParameter("postserver");
      postafsender = application.getInitParameter("postafsender");
    }
    if (drv==null) drv = "com.mysql.jdbc.Driver";
    if (url==null) url = "jdbc:mysql:///test";
    if (bru==null) bru = "root";
    if (adg==null) adg = "";

    System.out.println("url="+url+" bru="+bru+" adg="+adg); // til fejlfinding
    Class.forName(drv);                                    // indlæs driver
    con = DriverManager.getConnection(url, bru, adg);    // opret forbindelse

    Statement s = con.createStatement();

    try { // opret tabellen (dette giver en fejl hvis den allerede eksisterer)
      s.executeUpdate("CREATE TABLE brugere (brugernavn varchar(20), "+
                      "adgangskode varchar(20), epost varchar(50))");

      // Indsæt så også nogle brugere, så der er nogle til at starte med
      s.executeUpdate("INSERT INTO brugere VALUES ('Jacob','hemli','')");
      s.executeUpdate("INSERT INTO brugere VALUES ('Preben','hemli','')");
      s.executeUpdate("INSERT INTO brugere VALUES ('Søren','hemli','')");
      System.out.println("Bemærk: Standardbrugere oprettet (sikkerhedshul!)");
    } catch (SQLException sqlex) {} // OK med fejl her, log dem derfor ikke

    ResultSet rs = s.executeQuery("SELECT brugernavn FROM brugere");
    while (rs.next()) System.out.println( "bruger: "+rs.getString(1) );
    rs.close();
    s.close();
  }
  catch (Exception ex) {
    System.out.println("Problem med databasen: "+ex);
    ex.printStackTrace(); // Kritisk fejl, log den
    con = null; // sæt forbindelsen til null og prøv at oprette den senere
  }
}

/** Tjekker om brugernavn og adgangskode er OK */
public void tjekLogin()
{
  if (!tjek) return; // er der ikke sket ændringer behøver vi ikke tjekke igen
  loggetInd = false;
  tjek = false;
  if (brugernavn.length() > 0 && adgangskode.length() > 0) try {
    if (con == null) init(null); // ingen forbindelse − forsøg at oprette en
    PreparedStatement s = con.prepareStatement(
      "SELECT brugernavn FROM brugere WHERE brugernavn=? AND adgangskode=?");
    s.setString(1, brugernavn);
    s.setString(2, adgangskode);
    ResultSet rs = s.executeQuery();
    if (rs.next()) loggetInd = true; // korrekt! Brugeren er logget ind.
    else meddelelse = "Forkert brugernavn eller adgangskode";

    rs.close();
    s.close();
  } catch (Exception e) {
    e.printStackTrace();
```

```
      meddelelse = "Kunne ikke logge ind: " + e;
    }
  }

  /**
   * Registrerer ny bruger og sender adgangskode til epost-adressen
   * @return true hvis oprettelsen gik godt, ellers false.
   */
  public boolean opretBruger()
  {
    tjek = false;
    loggetInd = false;
    if (brugernavn.length() >= 1 && epost.length() >= 5 ) try {
      String nyAdgangskode = lavNyAdgangskode();
      PreparedStatement s = con.prepareStatement(
        "INSERT INTO brugere VALUES (?,?,?)");
      s.setString(1, brugernavn);
      s.setString(2, nyAdgangskode);
      s.setString(3, epost);
      s.executeUpdate();
      s.close();
      sendAdgangskode(nyAdgangskode);
      return true;              // oprettelse lykkedes!
    } catch (Exception e) {
      e.printStackTrace();
      meddelelse = "Fejl under oprettelsen: "+e;
    } else {
      meddelelse = "Brugernavn og epost skal være udfyldt";
    }
    return false;
  }

  /**
   * adgangskode genereres til ny bruger
   * @return koden
   */
  private String lavNyAdgangskode()
  {
    String ord = "";
    for (int j=0; j<6; j++) {
      ord = ord + (char) ('a' + (char) (Math.random()*25));
    }
    return ord;
  }

  /**
   * Send adgangskoden til brugeren med e-post
   */
  private void sendAdgangskode(String adgangskoden) throws Exception
  {
    if (postafsender==null) postafsender = "din@adresse.dk";
    if (postserver==null) postserver = "post.tele.dk";

    Properties prop = new Properties();
    prop.setProperty("mail.host", postserver);   // afhænger af internetudbyder
    prop.setProperty("mail.transport.protocol", "smtp");
    Session session = Session.getInstance(prop);

    // Opbyg beskedden
    Message besked = new MimeMessage(session);
    besked.setFrom(new InternetAddress(postafsender));
    besked.setRecipient(Message.RecipientType.TO, new InternetAddress(epost));
    besked.setSubject("Din adgangskode til javabog.dk");
    String txt = "Din adgangskode til javabog.dk er: "+adgangskoden+"\n"
      + "Du kan også logge ind ved at klikke på nedenstående henvisning:\n"
      + "http://javabog.dk:8080/JSP/kode/kapitel_09/log_ind.jsp?brugernavn="
      + java.net.URLEncoder.encode(brugernavn,"UTF-8")+"&adgangskode="
      + java.net.URLEncoder.encode(adgangskoden,"UTF-8")+"&handling=log+ind";
    besked.setContent(txt, "text/plain"); // put beskedden ind
    System.out.println(txt);              // skriv også beskedden i serverens log

    Transport.send(besked);               // send beskedden
    meddelelse = "Adgangskoden er sendt til adressen "+ epost;
  }
}
```

Metoden init() skal kaldes før javabønnen tages i brug. Den initialiserer forbindelsen til databasen (og opretter for nemheds skyld bruger-tabellen, hvis den ikke allerede findes)[2].

Metoden forventer at application-objektet bliver overført, sådan at den kan aflæse initialiseringsparametrene dbDriver, dbUrl, dbBruger, dbAdgangskode fra web.xml, som beskrevet i afsnit 4.5.5 (der er dog nogle fornuftige standardværdier i koden, hvis application-objektet ikke overføres eller initialiseringsparametrene ikke er defineret).

Metoden tjekLogin() tjekker, ved at kontakte databasen, om brugeren har angivet korrekt brugernavn og adgangskode. Af sikkerhedsgrunde bruger den PreparedStatement, for at undgå SQL-injektioner (se afsnit 5.4.2).

Resultatet kan aflæses med metoden isLoggetInd(), der svarer til egenskaben *loggetInd*.

Metoden sendAdgangskode() bruger JavaMail til at sende adgangskoden til brugeren med epost. For at det virker i din webapplikation skal du have sat initialiseringsparameteren mailserver til at pege på din internetudbyders SMTP−server.

9.5.3 Login−siden

Her kommer log_ind.jsp, skærmbilledet, hvor brugeren kan logge ind:

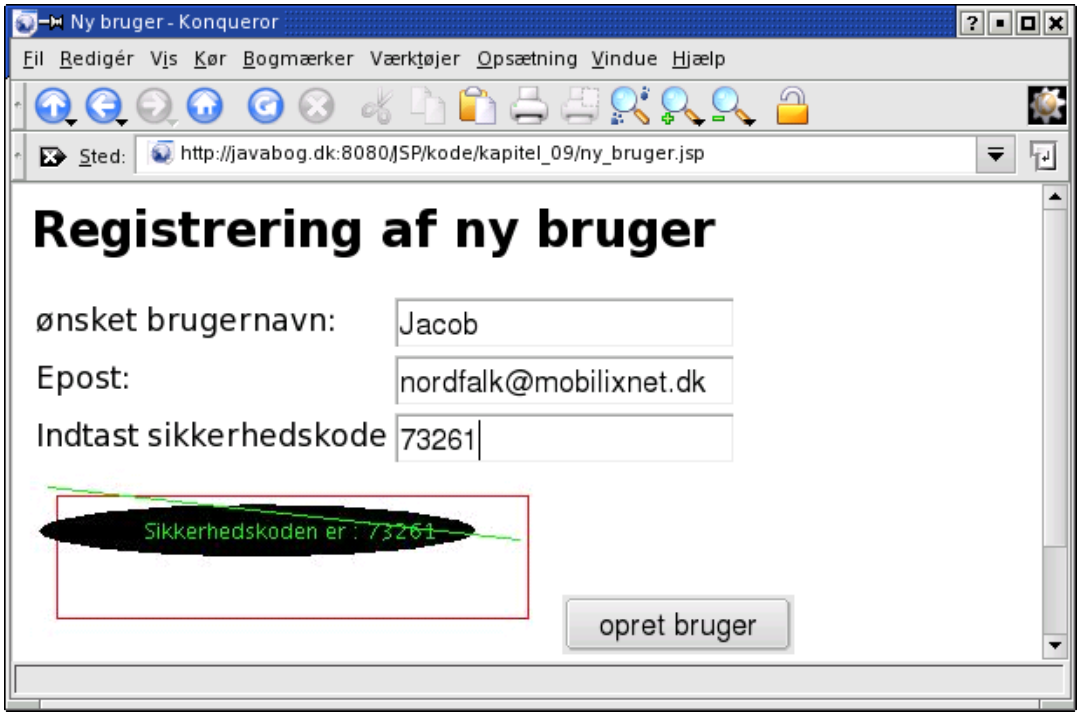
log_ind.jsp

```
<jsp:useBean id="login" class="javabog.Login" scope="session">
  <% login.init(application); %>  <%-- køres første gang bønnen bruges --%>
</jsp:useBean>

<%
  // Hvis brugernavn og kode er sat i en cookie (se afsnit 3.6.5) så brug dem:
  Cookie[] cookier = request.getCookies();
  if (cookier != null) for (int i=0; i<cookier.length; i++) {
    Cookie c = cookier[i];
    System.out.println("cookie "+c.getName()+"="+c.getValue());
    if (c.getName().equals("brugernavn")) login.setBrugernavn(c.getValue());
    if (c.getName().equals("adgangskode")) login.setAdgangskode(c.getValue());
  }
  // Hvis brugernavn og kode kommer med request-objektet så sæt dem:
%>
<jsp:setProperty name="login" property="brugernavn"/>
<jsp:setProperty name="login" property="adgangskode"/>

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

<img src="../kapitel_02/billede_med_avatar.jsp" align="right">
<h1>Log ind</h1>

<form method="post" action="log_ind.jsp">
<table>
<tr>
  <td>Brugernavn:</td>
  <td><input type="text" name="brugernavn" value="<%=login.getBrugernavn()%>"></td>
</tr>
<tr>
  <td>Adgangskode:</td>
  <td><input type="password" name="adgangskode"></td>
</tr>
</table>

<input type="submit" name="handling" value="log ind">
    <input type="checkbox" name="saet cookie">Husk mig på denne computer<br>
Jeg er <a href="ny_bruger.jsp">ny bruger</a> og ønsker at registrere mig.
</form>
<p>

<font color="red">
<%
login.tjekLogin();
if (login.isLoggetInd()) {
  if (request.getParameter("saet cookie") != null) {
    response.addCookie(
      new Cookie("brugernavn", request.getParameter("brugernavn")));
    response.addCookie(
```

```
      new Cookie("adgangskode", request.getParameter("adgangskode")));
  }
  %>
  Du er logget ind som <jsp:getProperty name="login" property="brugernavn"/>.
  <%
} else {
  String handling = request.getParameter("handling");
  if ("log ind".equals(handling)) {
    if (login.getBrugernavn().length()>0) {
      %>
      <%=login.getMeddelelse()%>
      Prøv igen.
  <%  } else { %>
      Indtast brugernavn og adgangskode.
      <%
    }
  }
}
%>
</font>

<a href="beskyttet_side.jsp">Gå til den beskyttede side</a>
</body>
</html>
```

9.5.4 Brugeroprettelsen

Hvis der er tale om en ny bruger vises dette skærmbillede, hvor brugeren kan vælge et brugernavn og angive sin epost−adresse, for at få oprettet en bruger.



For at undgå at alle mulige opretter brugere i flæng (eller endda laver et program, der gør det automatisk!), skal der også indtastes en sikkerhedskode, der bliver på et billede. Til det bruger vi eksemplet fra afsnit 2.8.5, til at producere et JPG−billede med sikkerhedskode. Teksten på billedet bestemmer vi ved at sætte attributten "billedtekst" i sessionen.

ny_bruger.jsp

```
<jsp:useBean id="login" class="javabog.Login" scope="session"/>
<jsp:setProperty name="login" property="brugernavn"/>
<jsp:setProperty name="login" property="epost"/>

<html>
<head><title>Ny bruger</title></head>
<body>

<h1>Registrering af ny bruger</h1>

<form action="ny_bruger.jsp" method="get">
<table>
<tr>
  <td>Ønsket brugernavn:</td>
  <td><input type="text" name="brugernavn" value="<%= login.getBrugernavn() %>">
  </td>
</tr>
<tr>
  <td>Epost:</td>
  <td><input type="text" name="epost" value="<%= login.getEpost() %>"></td>
```

```
</tr>
<tr>
  <td>Indtast sikkerhedskode</td>
  <td><input type="text" name="sikkerhedskode"></td>
</tr>
</table>

<% // Ekstra sikkerhedskode som bruger skal aflæse fra billede
  String sikkerhedskode = (String) session.getAttribute("sikkerhedskode");
  if (sikkerhedskode==null) {
    sikkerhedskode = "" + (int) (Math.random()*100000);
    session.setAttribute("sikkerhedskode", sikkerhedskode);
  }
  // Billedet henter sin tekst fra sessionsattribut "billedtekst", se afsnit 2.8.5
  session.setAttribute("billedtekst", "    Sikkerhedskoden er : "+sikkerhedskode);
%>
<img src="../kapitel_02/billede.jsp">
  <input type="submit" name="handling" value="opret bruger">
</form>

<font color="red">
<%
  String handling = request.getParameter("handling");
  if ("opret bruger".equals(handling)) {
    if (!sikkerhedskode.equals(request.getParameter("sikkerhedskode"))) {
      %>Du har tastet en forkert sikkerhedskode. <%
    } else   if (login.opretBruger()) {
      %><jsp:forward page="log_ind.jsp" /><%
    } else {
      %>Bruger kunne ikke oprettes. Prøv med et andet brugernavn og tjek epost.<%
    }
  }
%>
<%= login.getMeddelelse() %>
</font>

<br>
<a href="log_ind.jsp">Gå til login</a>
</body>
</html>
```

9.5.5 En beskyttet side

Den beskyttede side tjekker om der er logget korrekt ind:

beskyttet_side.jsp

```
<jsp:useBean id="login" class="javabog.Login" scope="session"/>

<%-- klient-omdirigering med JSP --%>
<% if (!login.isLoggetInd()) response.sendRedirect("log_ind.jsp"); %>


<%-- server-omdirigering med JSP
<% if (!login.isLoggetInd()) { %><jsp:forward page="log_ind.jsp"/><% } %>
--%>

<%-- server-omdirigering med JSTL
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:if test="${!login.loggetInd}"><jsp:forward page="log_ind.jsp" /></c:if>
--%>

<html>
<head><title>Beskyttet side</title></head>
<body bgcolor="#ffffff">

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

</body>
</html>
```

Bemærk at det kunne være en god idé at lægge login-tjekket i et kodefragment, som beskrevet i afsnit 4.2.1. I eksemplet er brugt klient-omdirigering (kommenteret bort findes en server-omdirigering - se afsnit 4.3).

9.5.6 Før eksemplet kan køre på din egen server

Eksemplet bruger klasser fra pakken javax.mail til at sende adgangskoden til brugeren samt MySQL-klasser til at kommunikere med en database.

Disse klasser findes ikke som standard i Tomcat, men de kan nemt installeres, ved at kopiere nogle JAR-filer ind i din webapplikations WEB-INF/lib/ eller Tomcats common/lib/.

For at bruge javax.mail, skal du kopiere filerne mail.jar og activation.jar derind. Søg efter filerne på din computer, de er nemlig sandsynligvis allerede inkluderet i dit udviklingsværktøj. Ellers hent dem fra http://java.sun.com/products/javamail (bemærk at du også skal hente activation.jar fra http://java.sun.com/products/javabeans/glasgow/jaf.html).

Hvordan du får fat i databasedriverklasserne er beskrevet i f.eks. afsnit 5.3.2, Kontakt til MySQL−database. Kopiér også denne JAR−fil ind.

Se afsnit 4.9.6, Hvis klasse(bibliotek)er ikke kan findes, hvis du har problemer med at få webserveren til at finde JAR−filerne.

9.5.7 Opgaver

1. Afprøv eksemplet, først ved at se det på: http://javabog.dk:8080/JSP/kode/, dernæst i din egen webserver (du skal måske rette i databasedriverklassen og database−URLen).Husk at du skal bruge en række ekstra klasser (se afsnit 9.5.6).
2. Som koden foreligger, kan enhver registrere sig og logge ind. Du skal derfor ændre (begrænse) systemet, sådan at brugere ikke selv kan registrere sig, men skal være oprettet i forvejen.
3. Du skal ændre kalender−eksemplet, sådan at brugerne af kalenderen skal være logget ind med brugernavn og adgangskode før de kan få adgang til kalenderen.
4. Prøv at skrive login−eksemplet om, sådan at brugere godt nok kan registrere sig, men ikke får adgang til de beskyttede sider, før de er blevet godkendt af en administrator.

9.6 Test dig selv

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

9.7 Resumé

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

1Hvis en klasse skal kunne bruges andre steder fra (f.eks. fra en JSP−side) skal den kunne importeres. Det kan klasser uden pakkenavn ikke fra og med JDK 1.4.

2Bruger du MySQL og får fejlen 'Server configuration denies access to data source', så kig på http://dev.mysql.com/doc/connector/j/en/cj−troubleshooting.html for enløsning.

10 Arkitekturer i webprogrammering

Dette kapitel er frivillig læsning; det forudsættes ikke i resten af bogen. Det forudsætter kapitel 5, Brug af databaser, og kapitel 9, Javabønner i JSP−sider.

Efterhånden som en webapplikation vokser sig større og større, bliver den sværere at overskue. Det bliver svært at finde rundt i de mange linjers kode i de mange filer og fejlfinding og videreudvikling bliver mere og mere tidskrævende.

Der opstår et behov for et system, der opdeler applikationens dele i nogle bidder, der hver i sær har et klart ansvar og som er nogenlunde afgrænset fra de andre dele af applikationen.

10.1 Trelagsmodellen

Et hyppigt brugt system til inddeling er at *lagdele* applikationen, oftest i tre lag:

- Øverste lag er JSP−siderne med HTML−koden og præsentationslogik.
- Mellemste lag er klasser, der tager sig af forretningslogikken i applikationen.
- Nederste lag er databasen.

Denne − nærmest klassiske − opdeling er meget populær og kaldes *trelagsmodellen*. JSP−siderne har ikke nogen databasekode selv, men kalder metoder i forretningslogik−klasser, som så fremfinder og ændrer på de relevante data i databasen. Brugerinteraktions−kode (behandle parametre fra request−objektet og beslutning om, hvilken side brugeren skal se næste gang) ligger som regel i øverste lag.

Trelagsmodellen er en god løsning i mange sammenhænge, men i en webapplikation kan øverste lag (JSP−siderne) vokse sig endog meget stort og en yderligere opdeling derfor blive nødvendig.

Model−View−Controller−arkitekturen er et meget populært bud på hvordan trelagsmodellen yderligere kan inddeles. En stor del af kapitlet handler derfor om denne arkitektur.

10.2 Model 1 og model 2−arkitekturer

Inden for webprogrammering taler man ikke om trelagsmodellen men om model 1 og model 2−arkitekturer.

10.2.1 Model 1−arkitektur: Logik sammen med HTML

I webprogrammeringens spæde barndom lavede de fleste mindre applikationer, hvor programlogikken lå sammen med HTML−koden. Et eksempel på denne måde at programmere kan findes i afsnit 5.5 (Eksempel − gæstebog).

Har man en webapplikation hvor programlogikken ligger sammen med HTML−koden kaldes det en model 1−arkitektur. Denne arkitektur har følgende konsekvenser:

- 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)

10.2.2 Model 2−arkitektur: Logik adskilt fra HTML

Efterhånden som webprogrammerne blev større og større og krævede mere og mere vedligeholdelse, blev følgende anbefaling mere og mere relevant:

> 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 kan koncentrere sig om funktionalitet og den bagvedliggende kode, som behandler data og afgør, hvilken side der skal vises

Dette kaldes en model 2−arkitektur. Denne arkitektur har følgende konsekvenser:

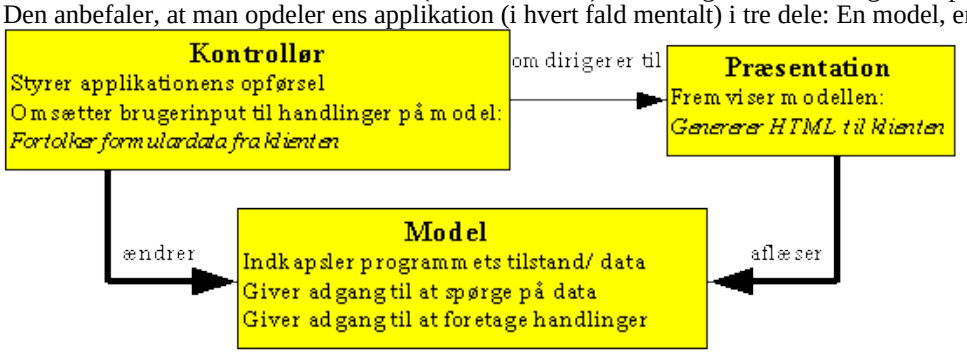
- Mere omfattende struktur
- Sværere at starte med
- Lettere at vedligeholde ved større projekter
- Programlogik og HTML relativt adskilt
- Forskellige personer kan tage sig af programmering og HTML−design
- Centraliseret: Programlogik og beslutning om, hvilken side, der skal vises sker ét sted

En udbygning af tankegangen om at adskille tekstligt indhold og programkode er også, at adskille sprogligt indhold fra HTML−design, f.eks. for at kunne lade webapplikationen kunne køre på flere sprog (internationalisering). Dette behandles i kapitel 13, Internationale sider.

Model 2−arkitekturen kaldes også Model−View−Controller−arkitekturen. I næste afsnit vil vi gå i dybden med delene i teorien og i det efterfølgende afsnit give et konkret eksempel.

10.3 Model−View−Controller−arkitekturen

Model−View−Controller−arkitekturen (forkortet MVC) er et designmønster beregnet til programmer med en brugergrænseflade. Den anbefaler, at man opdeler ens applikation (i hvert fald mentalt) i tre dele: En model, en præsentation og en kontrollør:



Model−View−Controller−arkitekturen og det meste af ovenstående diagram er ikke specielt rettet mod webapplikationer, men gælder for alle slags programmer. På ovenstående diagram er forklaringer, der er specielt rettet mod webapplikationer, skrevet i kursiv.

Lad os nu kigge nærmere på de tre dele af Model−View−Controller−arkitekturen.

10.3.1 Modellen

Modellen *indeholder data og registrerer, hvilken tilstand den pågældende del af programmet er i*

Oftest er data indkapslet i nogle klasser, sådan at konsistens sikres. I så fald er der kun adgang til at spørge og ændre på data gennem metodekald.

Modellen bør være uafhængig af, hvordan data præsenteres over for brugeren og er der flere programmer, der arbejder med de samme slags data, kan de i princippet have den samme datamodel, selvom de i øvrigt er helt forskellige.

Eksempel: En bankkonto har navn på ejer, kontonummer, kort−ID, saldo, bevægelser, renteoplysninger e.t.c.. Saldoen kan ikke ændres direkte, men med handlingerne overførsel, udbetaling og indbetaling kan saldoen påvirkes (se eksempelvis klassen Kontomodel i afsnit 10.4.1).

Bemærk hvordan modellen for en bankkonto er universel. Modellen kunne anvendes f.eks. både i et program til en Dankort−automat, et hjemmebank−system og i programmet som kassedamen anvender ved skranken.

Den samme model kunne anvendes både af en webapplikation og et almindeligt program.

10.3.2 Præsentationen

Præsentationen (eng.: View) *henter relevante data fra modellen og viser dem for brugeren i en passende form*

Selvom to præsentationer deler model (viser data fra samme model) kan de være meget forskellige, da de er beregnet på en bestemt brugergrænseflade.

Eksempel: Bankkontoen præsenteres meget forskelligt. I en Dankort−automat vises ingen personlige oplysninger overhovedet. I et hjemmebank−system kan saldo og bevægelser ses. Ved skranken kan medarbejderen se endnu mere, f.eks. filial og kontaktperson i banken (det kunne være implementeret som en grafisk applikation, der kører hos brugeren).

I en webapplikation producerer præsentationen HTML−koden, som brugeren ser. Oftest er præsentationen implementeret som nogle JSP−sider (nogle kan dog foretrække servletter).

10.3.3 Kontrolløren

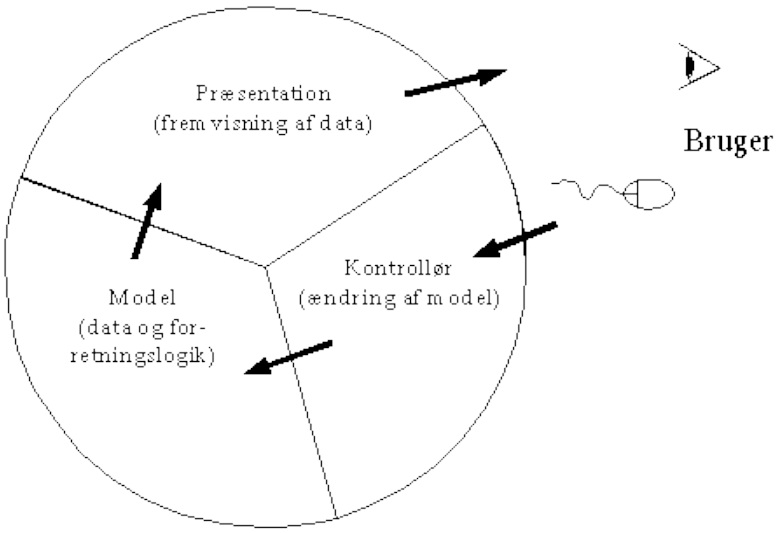
Kontrolløren (eng.: Controller) definerer hvad programmet kan. Den *omsætter brugerens indtastninger til handlinger, der skal udføres på modellen*

Eksempel: I Dankort−automaten kan man kun hæve penge. I et hjemmebank−system kan brugeren måske lave visse former for overførsel fra sin egen konto. Ved skranken kan medarbejderen derudover foretage ind− og udbetalinger.

I en webapplikation er kontrolløren den, der modtager alle parametre fra formularer fra klienten og derefter beslutter, hvad der skal ske. Oftest er præsentationen implementeret som en eller flere servletter (nogle kan dog foretrække at bruge JSP−sider).

10.3.4 Informationsstrøm gennem MVC

Figuren herunder illustrerer hvordan strømmen af information går fra modellen, via præsentationen til brugeren (symboliseret ved et øje).



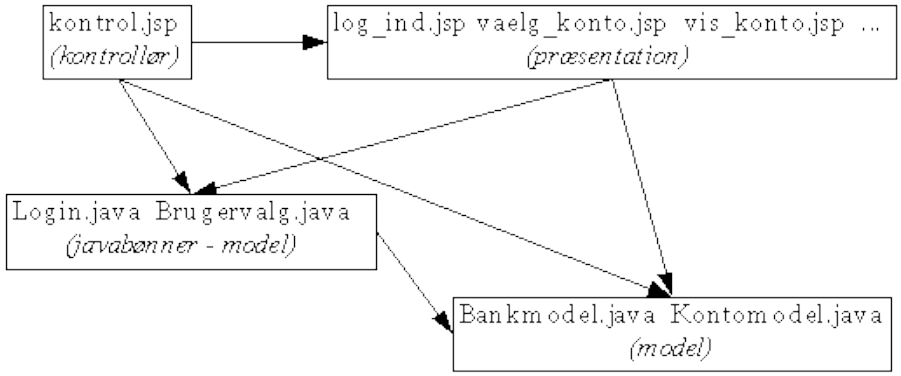
Brugeren foretager nogle handlinger (symboliseret ved musen), som via kontrolløren fortolkes som nogle ændringer, der foretages på modellen, hvorefter de nye data vises for brugeren.

I praksis foregår det i en webapplikation ved, at brugeren udfylder en formular, som sendes til serveren, hvor de modtages af kontrolløren. Kontrolløren fortolker parametrene og beslutter, hvad det er brugeren vil gøre og kalder de tilsvarende metoder på modellen. Til sidst omdirigerer kontrolløren til en præsentations−side med HTML−koden brugeren skal se.

10.4 Eksempel på MVC: Bankkonti

Det følgende er et eksempel på hvordan en MVC−arkitektur kunne implementeres.

Det består af følgende JSP−sider og klasser:



Pilene viser hvilke sider/klasser, der kender til andre sider/klasser.

Kontrolløren er her lavet som JSP−siden kontrol.jsp. Den modtager alle forespørgsler, ændrer i objekterne og omdirigerer til en af de andre sider. De andre JSP−sider præsenterer data, som de aflæser fra objekterne og kender ikke til kontrolløren.

Modellen består af forretningslogikobjekterne Bankmodel og Kontomodel. Hertil kommer Javabønnerne Login og Brugervalg, som husker og behandler data nærmere præsentationslaget[1]. Der findes ét Brugervalg− og Login−objekt per bruger (knyttet til brugerens session).

I det følgende gennemgås koden i rækkefølgen Model, Præsentation (d.v.s. de JSP−sider der producerer HTML) og til sidst kontrolløren.

10.4.1 Model (Bankmodel.java og Kontomodel.java)

Vi har brug for en liste over samtlige konti i banken. Denne klasse kaldes Bankmodel:

Bankmodel.java

```
package bank;

import java.util.*;

public class Bankmodel
{
  public List konti = new ArrayList();

  public Bankmodel()
  {
    konti.add(new Kontomodel("Jacob",  1001, 113.75));
    konti.add(new Kontomodel("Jacob",  1002, 555.00));
    konti.add(new Kontomodel("Preben", 1101,   8.50));
    konti.add(new Kontomodel("Søren",  1201,  78.25));
  }
}
```

Bankmodellen burde selvfølgelig, i et mere realistisk eksempel, have sine data i en database, men for letheds skyld opretter den her blot en fast liste over konti.

Hver konto er repræsenteret af et Kontomodel−objekt, med ejer, id (kontonummer), saldo og en liste over de bevægelser, der har været på kontoen.

Kontomodel.java

```
package bank;

import java.util.*;

public class Kontomodel
{
  private String ejer;
  private int id;
  private double saldo;
  private List bevægelser = new ArrayList();

  public Kontomodel(String ejer1, int id1) { ejer = ejer1; id = id1; }
  public Kontomodel(String ejer1, int id1, double saldo1)
    { ejer = ejer1; id = id1; saldo = saldo1; }

  public String getEjer()     { return ejer; }
  public int getId()          { return id; }
```

```
  public double getSaldo()    { return saldo; }
  public List getBevægelser() { return bevægelser; }

  public String toString()    { return ejer + ": "+saldo+" kr"; }

  public void overfør(Kontomodel til, double beløb)
  {
    if (beløb<=0) throw new IllegalArgumentException("Beløb skal være positivt");
    if (beløb>saldo) throw new IllegalArgumentException("Der er ikke penge nok");
    saldo = saldo - beløb;
    til.saldo = til.saldo + beløb;// privat variabel kan ændres i samme klasse

    String ændring = "Overført "+beløb+" fra "+ejer+" til "+til.ejer;
    bevægelser.add(ændring);
    til.bevægelser.add(ændring);
  }

  public void hæv(double beløb)
  {
    if (beløb<=0) throw new IllegalArgumentException("Beløb skal være positivt");
    if (beløb>saldo) throw new IllegalArgumentException("Der er ikke penge nok");
    saldo = saldo - beløb;
    bevægelser.add("Hævet "+beløb);
  }

  public void indsæt(double beløb)
  {
    if (beløb<=0) throw new IllegalArgumentException("Beløb skal være positivt");
    saldo = saldo + beløb;
    bevægelser.add("Indsat "+beløb);
  }
}
```

Bemærk hvordan klassen kun koncentrerer sig om forretningslogik.

Opstår der en fejlsituation i en metode, kastes en undtagelse, der beskriver fejlen, sådan at kalderen kan håndtere den (se hvordan senere, i afsnit 10.4.5, Fejlhåndtering).

10.4.2 Javabønnen (Brugervalg.java)

Her er Brugervalg−klassen. Metoderne, der svarer til egenskaber, er fremhævet med fed.

Brugervalg.java

```
package bank;
import java.util.*;
public class Brugervalg
{
  /** Egenskab 'bankmodel' sættes fra JSP−siden kontrol.jsp når bønnen oprettes */
  Bankmodel bank;
  public void setBankmodel(Bankmodel b) { bank = b; }

  /** Liste over denne brugers konti */
  public ArrayList konti;

  /** Den konto, brugeren har valgt at arbejde med lige nu */
  public Kontomodel konto;

  /** Egenskab 'kontovalg' sættes fra JSP−side vaelg_konto.jsp*/
  public void setKontovalg(int nr) {
    konto = null;
    for (int i=0; i<konti.size(); i++) {
      Kontomodel k = (Kontomodel) konti.get(i);
      if (k.getId() == nr) konto = k;
    }
    if (konto==null) throw new IllegalArgumentException("Ukendt Konto−ID: "+nr);
  }

  /** Streng der beskriver en handling brugeren ønsker at udføre */
  public String handling;
  public void setHandling(String h) { handling = h; }

  /** Beløbet handlingen (hæv/sæt ind/overfør) drejer sig om */
  double handlBeløb;
  public void setBeloeb(double b) { handlBeløb = b; }

  /** Hvis der skal foretages en overførsel, hvilken konto er det til */
  Kontomodel ovfTil;

  /** Egenskab 'tilKontoId' sættes fra JSP−siden vis_konto.jsp */
  public void setTilKontoId(int nr) {
    ovfTil = null;
    for (int i=0; i<bank.konti.size(); i++) {
      Kontomodel k = (Kontomodel) bank.konti.get(i);
      if (k.getId() == nr) ovfTil = k;
    }
    if (ovfTil==null) throw new IllegalArgumentException("Ukendt Konto−ID: "+nr);
  }
```

```
  public void udførHandling()
  {
    try {
      if (handling.equals("haev"))          konto.hæv(handlBeløb);
      else if (handling.equals("saet_ind")) konto.indsæt(handlBeløb);
      else if (handling.equals("overfoer")) konto.overfør(ovfTil, handlBeløb);
      else System.out.println("Ukendt handling: "+handling);
    } finally {
      handling = null; // selv hvis noget gik helt galt skal data nulstilles
      handlBeløb = 0;
      ovfTil = null;
    }
  }
}
```

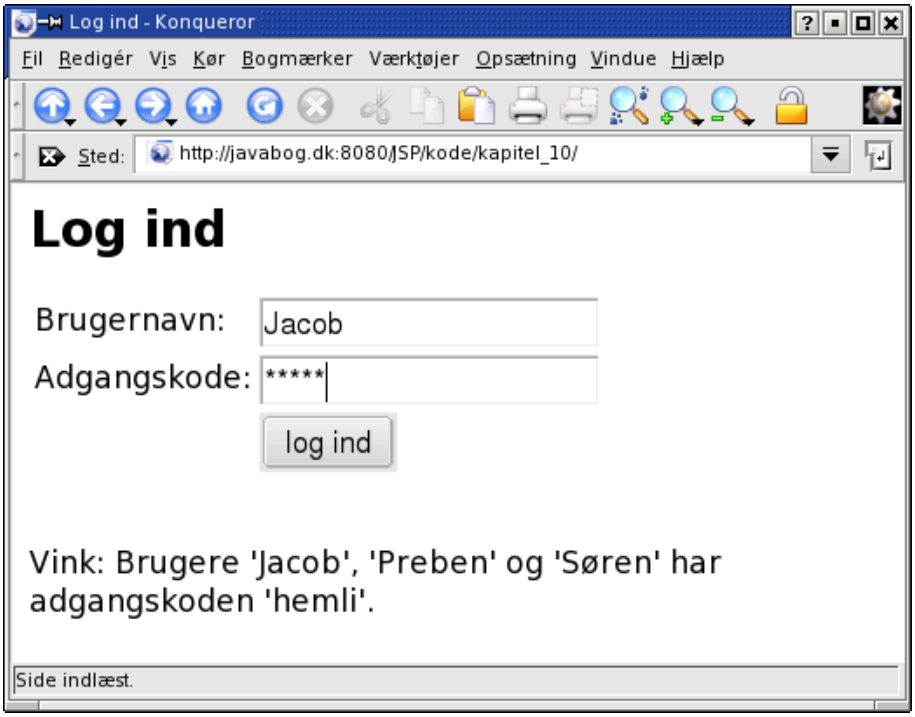
En udskrift af Login−bønnen kan ses i afsnit 9.5.2.

10.4.3 Præsentationen (JSP−siderne)

Først log_ind.jsp, hvor brugeren kan logge ind. Den genbruger Login−bønnen fra afsnit 9.5:

log_ind.jsp

```
<jsp:useBean id="login" class="javabog.Login" scope="session"/>
<jsp:setProperty name="login" property="*"/>
<html>
<head><title>Log ind</title></head>
<body>

<h1>Log ind</h1>

<form>
<table>
<tr>
  <td>Brugernavn:</td>
  <td><input name="brugernavn" type="text"
             value='<jsp:getProperty name="login" property="brugernavn"/>'></td>
</tr>
<tr>
  <td>Adgangskode:</td>
  <td><input type="password" name="adgangskode"></td>
</tr>
<tr>
  <td></td>
  <td><input type="submit" name="handling" value="log ind"></td>
</tr>
</table>
</form>

<p><font color="red"><%= login.getMeddelelse() %></font><br>
Vink: Brugere 'Jacob', 'Preben' og 'Søren' har adgangskoden 'hemli'.</p>

</body>
</html>
```



Når brugeren først er logget ind skal han, hvis han har flere konti, vælge hvilken konto han vil arbejde med. Det sker i vaelg_konto.jsp:

vaelg_konto.jsp

```
<jsp:useBean id="valg" class="bank.Brugervalg" scope="session" />
<html>
<head><title>Vælg konto</title></head>
<body>
<h1>Vælg hvilken konto du vil bruge</h1>

<form>

<%
  for (int i=0; i<valg.konti.size(); i++) {
    bank.Kontomodel k = (bank.Kontomodel) valg.konti.get(i);
    %>
      <input type="radio" name="kontovalg" value="<%= k.getId() %>">
      Konto <%= k.getId() %> med <%= k.getSaldo() %> kr.<br>
    <%
  }
%>
<input type="submit" value="Vælg konto">
</form>

<p><a href="?handling=log_ud">Log ud</a>
</body>

</html>
```

Når man har valgt en konto, kommer man videre til vis_konto.jsp:

vis_konto.jsp

```
<%@ page import="java.util.*" %>
<jsp:useBean id="valg" class="bank.Brugervalg" scope="session" />
<html>
<head><title>Vis konto</title></head>
<body>
<h1>Konto <%= valg.konto.getId() %> med ejer <%= valg.konto.getEjer() %></h1>
Saldo: <%= valg.konto.getSaldo() %><p>
Bevægelser:<br>
<%
  List bev = valg.konto.getBevægelser();
  for (int i=0; i<bev.size(); i++) {
    %>
      <%= bev.get(i) %><br>
    <%
  }
%>
<hr>
Hvad vil du gøre:<p>

<form>
  <input type="hidden" name="handling" value="haev">
  Hæv <input type="text" name="beloeb" size="4"> kr. fra kontoen.
  <input type="submit" value="Hæv">
</form>
<p>
```

```
<form>
  <input type="hidden" name="handling" value="saet_ind">
  Sæt <input type="text" name="beloeb" size="4"> kr. ind på kontoen.
  <input type="submit" value="Sæt ind">
</form>
<p>

<form>
  <input type="hidden" name="handling" value="overfoer">
  Overfør <input type="text" name="beloeb" size="4"> kr.
  til konto nr. <input type="text" name="tilKontoId" size="6">.
  <input type="submit" value="Overfør">
</form>

<p><a href="?handling=log_ud">Log ud</a>
</body>
</html>
```

Hvis brugeren ikke har nogen konti tilknyttet, kommer han til ingen_konto.jsp i stedet for vaelg_konto.jsp:

ingen_konto.jsp

```
<jsp:useBean id="login" class="javabog.Login" scope="session"/>
<html>
<head><title>Ingen konto</title></head>
<body>

<h1>Ingen konti fundet.</h1>
Brugeren <%= login.getBrugernavn() %> har ingen konti tilknyttet.<p>
Kontakt venligst banken for at få oprettet en konto.

<p><a href="?handling=log_ud">Log ud</a>
</body>
</html>
```

10.4.4 Kontrolløren (kontrol.jsp)

Her kommer kontrol.jsp, der sørger for kontrollør−delen af applikationen: At manipulere med modellen (bønnernes egenskaber oftest) ud fra brugerens valg (den indkomne formular) samt at beslutte, hvilken side der derefter skal vises.

Den er lavet som en JSP−fil, men producerer ikke selv noget output (den omdirigerer i stedet til præsentations−JSP−filerne vist tidligere).

kontrol.jsp

```
<%-- Hvis der opstår en undtagelse omdirigeres automatisk til fejl.jsp --%>
<%@ page contentType="text/html; charset=iso-8859-1" errorPage="fejl.jsp"
         import="java.util.*, bank.*" %>

<jsp:useBean id="login" class="javabog.Login" scope="session">
  <% login.init(application); %>  <%-- køres første gang bønnen bruges --%>
</jsp:useBean>
<jsp:setProperty name="login" property="*" />

<%
  String handling = request.getParameter("handling");
  String brugernavn = login.getBrugernavn();  //<title>kontrol</title>

  if ("log ind".equals(handling)) login.tjekLogin();
  if ("log_ud".equals(handling))  login.setAdgangskode("");

  if (!login.isLoggetInd()) {            // er brugeren logget korrekt ind?
    application.log("Bruger "+brugernavn+" skal logge ind.");
    session.removeAttribute("valg");     // eller evt: session.invalidate()
    request.getRequestDispatcher("log_ind.jsp").forward(request,response);
    return;                              // afslut behandlingen af denne side
  }
%>

<%-- Nedenstående objekt har globalt virkefelt (svarer til en singleton) --%>
<jsp:useBean id="bank" class="bank.Bankmodel" scope="application" />

<jsp:useBean id="valg" class="bank.Brugervalg" scope="session">
  <% valg.setBankmodel(bank); %>  <%-- sæt bankmodel første gang --%>
</jsp:useBean>
<jsp:setProperty name="valg" property="*" />

<%
  if (valg.konto == null) {              // har han valgt en af sine konti?
    // nej - find alle brugerens konti og læg dem i valg.konti
    if (valg.konti == null) {
      valg.konti = new ArrayList();
      for (int i=0; i<bank.konti.size(); i++) {
        Kontomodel k = (Kontomodel) bank.konti.get(i);
        if (k.getEjer().equalsIgnoreCase(brugernavn)) valg.konti.add(k);
      }
    }

    if (valg.konti.size() == 0) {
      application.log(brugernavn+" skal oprette konto.");
      request.getRequestDispatcher("ingen_konto.jsp").forward(request,response);
    } else if (valg.konti.size() == 1) {
      application.log(brugernavn+" har kun en konto, den vælger vi!");
      valg.konto = (Kontomodel) valg.konti.get(0);
      request.getRequestDispatcher("vis_konto.jsp").forward(request,response);
    } else {
      application.log(brugernavn+" vælger ml. "+valg.konti.size()+" konti");
      request.getRequestDispatcher("vaelg_konto.jsp").forward(request,response);
    }
    return;                              // afslut behandlingen af denne side
  }

  if (valg.handling != null) {           // konto er valgt - nogen handlinger?
    application.log(brugernavn+" udfører handling: "+valg.handling);
    valg.udførHandling();
  }
  request.getRequestDispatcher("vis_konto.jsp").forward(request,response);
%>
```

Vi kunne også have lavet kontrolløren som en servlet (se afsnit 7.1), men da kunne vi ikke udnytte <jsp:setProperty ... />−mekanismen til at sætte bønnernes egenskaber.

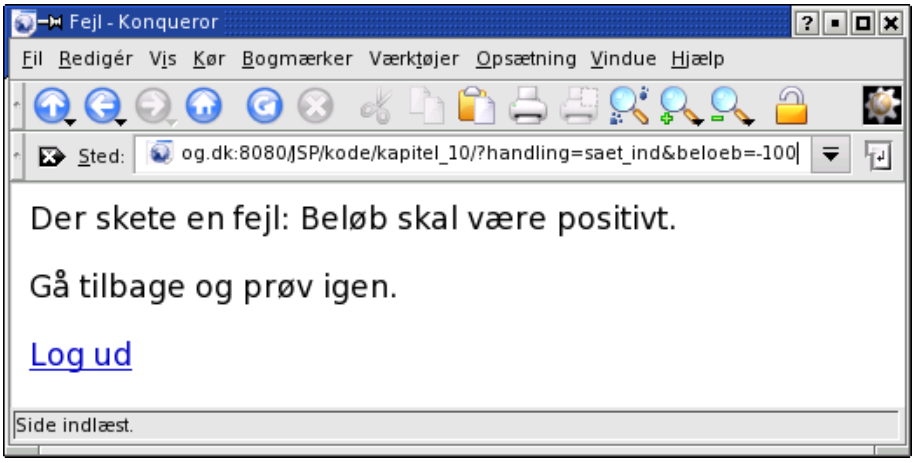
Derudover skal servletter ligge i en anden mappe, adskilt fra JSP−siderne, hvilket ville gøre det mere besværligt at prøve eksemplet.

Når brugervalg−bønnen oprettes, skal den initialiseres, ved at få kaldt setBankmodel() med bank−objektet. Det gør vi ved at lægge kode ind mellem <jsp:useBean> og </jsp:useBean>, som beskrevet i afsnit 9.2.7, Initialisering af javabønner:

```
<jsp:useBean id="valg" class="bank.Brugervalg" scope="session">
  <% valg.setBankmodel(bank); %>
</jsp:useBean>
```

10.4.5 Fejlhåndtering (fejl.jsp)

Indtaster brugeren noget forkert (f.eks. hæve et negativt beløb), opstår der en undtagelse (eng.: Exception) og siden fejl.jsp vises:



Dette sker fordi sidedirektivet errorPage="fejl.jsp" var sat i kontrol.jsp hvor fejl (undtagelser) kunne opstå:

```
<%-- Hvis der opstår en undtagelse omdirigeres automatisk til fejl.jsp --%>
<%@ page errorPage="fejl.jsp" %>
```

Hvis en undtagelse opstår, vil anmodningen derfor automatisk blive omdirigeret til siden fejl.jsp. Denne side skal have sat sidedirektivet isErrorPage="true" og har det implicitte objekt exception defineret:

fejl.jsp

```
<%@ page contentType="text/html; charset=iso-8859-1" isErrorPage="true" %>
<jsp:useBean id="login" class="javabog.Login" scope="session"/>
<html>
<head><title>Fejl</title></head>
<body>
Der skete en fejl: <%= exception.getLocalizedMessage() %>.
<p>
Gå tilbage og prøv igen.
<%
  application.log( (Exception)exception, "Fejl for "+login.getBrugernavn());
%>

<p><a href="?handling=log_ud">Log ud</a>
</body>
</html>
```

Se også afsnit 4.4 og afsnit 4.5.8 for information om fejlhåndtering.

10.4.6 Hvordan præsentation henviser til kontrollør

I præsentationssider har vi ikke angivet, hvor formulardata skal sendes hen. Der står blot:

```
<form>
```

og

```
<a href="?handling=log_ud">Log ud</a>
```

Vi *kunne* godt have skrevet kontrol.jsp ind i alle formularer og henvisninger, da det jo er den, der skal modtage data, f.eks.:

```
<form action="kontrol.jsp">
```

og

```
<a href="kontrol.jsp?handling=log_ud">
```

men det er strengt taget overflødigt, da vi bruger *serverside*-omdirigering (beskrevet i afsnit 4.3.2), sådan at netlæseren slet ikke ved, at der er omdirigeret til en anden side. Netlæseren indsender således formulardata til kontrol.jsp som den skal, selvom det i virkeligheden er en anden side på serveren, der har skabt formularen.

10.5 Designmønster: Frontkontrol

En Frontkontrol (eng.: Front Controller) sørger for at dirigere brugeren hen til den rigtige side. Hvis brugeren må navigere frit, vil frontkontrollen blot sørge for, at brugeren får de anspurgte sider. Hvis brugeren skal følge en bestemt vej gennem siderne, vil frontkontrollen tjekke brugerens indtastninger og omdirigere brugeren, sådan at vejen gennem siderne følges. Det kunne f.eks. være, at brugeren skulle indtaste brugernavn og adgangskode, for at kunne se nogle bestemte sider. Frontkontrollen ville da omdirigere brugere, der ikke er logget ind, til login-siden.

En Frontkontrol er altså en central del af webapplikationen, som alle anmodninger går igennem. Frontkontrollen beslutter, hvilken side der rent faktisk skal vises og omdirigerer til den.

Bankkonto−eksemplet

JSP−siden kontrol.jsp fra afsnit 10.4.4 er *næsten* en frontkontrol. Den er det ikke helt, fordi man kan kan rette i URLen og på den få fat i de andre sider uden om kontrol.jsp. Senere, i afsnit 10.5.3, vil vi se på, hvad der skulle gøres, for at brugeren ikke kan gå uden om den.

10.5.1 Variationer (til større applikationer)

I en større webapplikation bliver det besværligt, hvis én Frontkontrol skal stå for al omdirigering. Da kan man vælge, at kæde flere Frontkontroller, sådan at forskellige Frontkontroller kan stå for forskellige dele af applikationen.

Man kan også vælge, at Frontkontrollen kalder en eller flere eksterne klasser, der 'ekspederer' brugerens anmodninger og beslutter, hvad der skal ske i forskellige situationer (en ekspeditør − eng.: Dispatcher).

Skal applikationen kunne anvendes på flere forskellige sprog eller fra flere forskellige slags klienter, f.eks. både fra en almindelig netlæser (såsom Netscape/Mozilla) og fra PDA'er og mobiltelefoner, kunne man vælge at lade Frontkontrollen kalde en ekstern klasse (en 'View Mapper'), der beslutter, om klienten skal have svaret tilsendt som HTML eller som f.eks. WML.

10.5.2 Implementering af en Frontkontrol

En Frontkontrol er ofte implementeret som en servlet, men der er intet, der forhindrer en i at bruge en JSP−side som Frontkontrol i stedet. Det kan gøres på flere måder:

- Manuel inklusion af JSP−side, der agerer Frontkontrol: En simpel implementering af en Frontkontrol som en JSP−side kunne være, at alle JSP−sider inkluderede en bestemt anden JSP−side, der derfor blev udført som det første på hver side. Denne side kunne f.eks. tjekke om brugeren var logget ind og lave en omdirigering af klienten (se afsnit 4.3.1, Klient−omdirigering) til login−siden, hvis det ikke var tilfældet.
- Man redigerer web.xml sådan, at alle forespørgsler dirigeres til én servlet eller JSP−side, der agerer Frontkontrol. Den vælger derpå hvilken JSP−side, forespørgslen skal dirigeres videre til. Siderne ligger skjult for klienten i WEB−INF/−mappen og omdirigering skal derfor ske internt i serveren (se afsnit 4.3.2, Server−omdirigering).

Lad os se på den anden mulighed.

10.5.3 Eksempel på en Frontkontrol

Ved at rette i web.xml kan man tvinge al trafikken til én side (kontrol.jsp), der derpå vælger hvilken JSP−side, der skal vises. Her er den relevante del af web.xml:

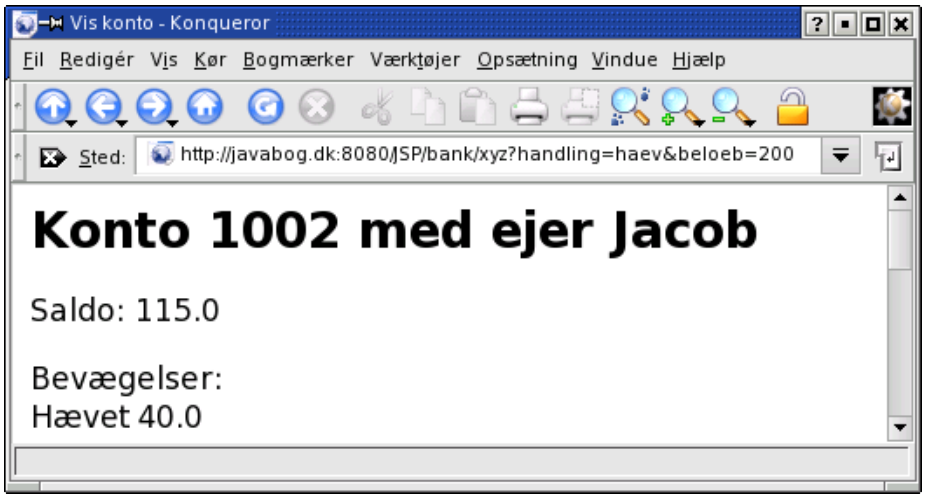
udsnit af WEB−INF/web.xml

```
<!−− Frontkontrol: Alt der starter med /bank sendes til kontrol.jsp −−>
<servlet>
  <servlet−name>Frontkontrol</servlet−name>
  <jsp−file>/WEB−INF/bank/kontrol.jsp</jsp−file>
</servlet>

<servlet−mapping>
  <servlet−name>Frontkontrol</servlet−name>
  <url−pattern>/bank/*</url−pattern>         <!−− Bemærk: * i URL−mønster −−>
</servlet−mapping>
```

Alle JSP−siderne flyttes nu ind i mappen WEB−INF/bank/, hvilket gør det umuligt for klienten at få fat i dem direkte.

Til gengæld angives, at alle forespørgsler, der passer med URL−mønstret /bank/*, skal dirigeres til frontkontrollen /WEB−INF/bank/kontrol.jsp, der dermed bliver den eneste side, brugerne får adgang til.

Lige meget hvilken URL brugeren skriver får han altså kontrol.jsp (her har en bruger forsøgt sig med bank/xyz som URL):



I omdirigeringen i kontrol.jsp skal den absolutte sti (/WEB−INF/bank/) nu angives, så:

```
request.getRequestDispatcher("log_ind.jsp").forward(request,response);
```

skal rettes til:

```
request.getRequestDispatcher("/WEB−INF/bank/log_ind.jsp").forward(request,response);
```

Øvelse

- Afprøv bankkonto−eksemplet. Tjek om du kan omgå kontrol.jsp og se nogle af de andre sider ved at rette i URLen. Ret f.eks. kode/kapitel_10/ til kode/kapitel_10/vaelg_konto.jsp
- Udfør de ovenstående ændringer og tjek om du stadig kan omgå kontrol.jsp. Ellers ret URLen fra kode/kapitel_10/ til bank/ (her er rettelserne allerede foretaget). Du kan også prøve eksemplet fra: http://javabog.dk:8080/JSP/bank/

10.6 Test dig selv

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

10.7 Resumé

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

10.8 Rammer for webarkitekturer

Der er rigtig mange rammer (eng.: frameworks) og standarder for hvordan en webapplikation med model 2−arkitektur skal bygges op. I det følgende vil de to vigtigste blive berørt.

10.8.1 Jakarta Struts

Struts er et Open Source−projekt baseret på JSP og servletter. Det understøtter Model−View−Controller−arkitekturen beskrevet i dette kapitel, på den måde at Struts udgør Frontkontrol (d.v.s. kontrolløren) og dele af præsentationen.

Der er under rammerne for Struts bl.a. mekanismer til implementering af indsamlingen af brugerinput fra formularer, validering af brugerinput, visning af meddelelser og fejltekster med understøttelse for forskellige sprog, opbygning af websider ud fra skabeloner m.v..

Struts er indenfor de seneste par år blevet meget populært og er nu en de facto standard inden for større webapplikationer. Alle udviklingsværktøjerne nævnt i denne bog understøtter således Struts.

10.8.2 JSF − Java Server Faces

Java Server Faces (JSF) er et system til at designe web−baserede brugergrænseflader i Java. Det er samme idé som bruges i WebObjects fra Apple og ASP.NET Web Forms fra Microsoft: Ligesom med almindelige grafiske brugergrænseflader i AWT eller Swing har man et sæt komponenter (knapper, indtastningsfelter, afkrydsningsfelter e.t.c. og det er også muligt at definere sine egne), som man bruger til at opbygge sin webside med.

Opbygningen kan ske i en visuel designer, som man kender det fra når man designer i AWT/Swing: Hver komponent er en javabønne, hvis egenskaber kan ændres (d.v.s. som kan aflæses/sættes med get− og set−metoder) og når brugeren foretager en handling, afsender komponenterne hændelser (eng.: events), som kan aktivere netop de stumper kode, der er brug for.

JSF giver en helt anden måde at gå til webprogrammering på: I stedet for at programmere JSP−sider, der undersøger request−objektet og producerer HTML, definerer man altså en JSF−side ved hjælp af komponenter og kan så manipulere med komponenterne og definere kode, der bliver udført når brugeren gør bestemte med komponenterne, f.eks. udfylder et indtastningsfelt. JSF sørger for det mellemliggende lag, med at undersøge request−objektet og producere den rigtige HTML.

JSFs arkitektur understøtter at komponenterne producerer forskellig HTML−kode afhængig af klientens formåen, sådan at samme JSF−applikation kan bruges til PDA'er, mobiltelefoner, almindelige netlæsere etc..

10.9 Mere læsning

Mere læsning om model 2−arkitektur og MVC

- Afsnit '4.3 Web−Tier Application Structure' og ' 11.1 J2EE Architecture Approaches' i http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/

Mere læsning om Struts

- Den officielle hjemmeside om Jakarta Struts: på http://struts.apache.org.

Mere læsning om Java Server Faces

- Suns officielle side om Java Server Faces:
  http://java.sun.com/j2ee/javaserverfaces/
- Om JSF i 'The J2EE Tutorial':
  http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JSFIntro.html
- Portal om JSF:
  http://www.jsfcentral.com
- Artikel: 'Developing Web Applications with JavaServer Faces':
  http://java.sun.com/developer/technicalArticles/GUI/JavaServerFaces/

Mere læsning om andre rammer for webarkitekturer

- Se under 'Servlets' og under 'Libraries and Frameworks' på
  http://directory.google.com/Top/Computers/Programming/Languages/Java/Server−Side/

1De kunne også opfattes som en del af kontrolløren.

11 XML, indholdssyndikering og webtjenester

Dette kapitel forudsætter kapitel 3, Interaktive sider. Det er kompakt skrevet og behandler en række avancerede emner, så lidt forhåndskendskab er en fordel.

11.1 Introduktion til XML

XML (eXtenisble Markup Language) minder meget om HTML, men har et andet formål:

- HTML er beregnet på *dokumenter der skal vises for et menneske* og det indeholder derfor både tekst og formateringskoder, der instruerer fremviseren (netlæseren) om, hvordan tekstindholdet skal vises.
- XML er beregnet til at *overføre data mellem programmer* − og det indeholder derfor (tekst−)data og koder, der instruerer programmet om, hvordan disse data skal fortolkes.

XML−teknologier har været i en rivende udvikling inden for de seneste par år og emnet kunne nemt fylde en separat bog. Af pladshensyn vil de følgende afsnit derfor blot definere de mest centrale begreber inden for XML og behandle de aspekter, der er relevant i forbindelse med JSP og webserverprogrammering.

11.1.1 XML−formatet

En simpel XML−fil kunne se således ud:

persongalleri.xml

```
<?xml version="1.0" encoding="iso−8859−1"?> <!−− filnavn: persongalleri.xml −−>
<galleri>
  <titel>Betydningsfulde personer</titel>

  <person id="1">
    <fornavn>Troels</fornavn>
    <efternavn>Nordfalk</efternavn>
    <fødselsdato år="1972" måned="08" dag="11" />
  </person>

  <person id="2">
    <fornavn>Jacob</fornavn>
    <efternavn>Nordfalk</efternavn>
    <fødselsdato år="1971" måned="01" dag="01" />
    <værker>
      <værk type="bog">
        <titel>Objektorienteret programmering i Java</titel>
        <isbn udgave="1">8779000940</isbn>
        <isbn udgave="2">8779001378</isbn>
      </værk>
      <værk type="hjemmeside">
        <titel>javabog.dk</titel>
        <url>http://javabog.dk</url>
      </værk>
      <værk type="bog">
        <titel>Videregående programmering i Java</titel>
        <isbn udgave="1">8779001955</isbn>
      </værk>
    </værker>
  </person>
</galleri>
```

En XML−fil er altså en tekstfil med data i nogle navngivne strukturer. Alle koder skal afsluttes igen med en slutkode, som i

```
    <fornavn>Jacob</fornavn>
```

er der intet mellem start− og slutkoden kan det forkortes ved at afslutte med />, f.eks.:

```
    <fødselsdato år="1971" måned="01" dag="01" />
```

Denne kode svarer præcist til:

```
    <fødselsdato år="1971" måned="01" dag="01"></fødselsdato>
```

En af de smarte ting ved XML−formatet (og HTML−formatet) i forhold til andre filformater er, at det kan udvides: Et program der f.eks ikke forstår afsnittet <værker> i ovenstående XML−fil kan nemt springe dette over. På denne måde kan et filformat senere udvides med flere data, uden at det påvirker eksisterende programmer, der bruger filformatet.

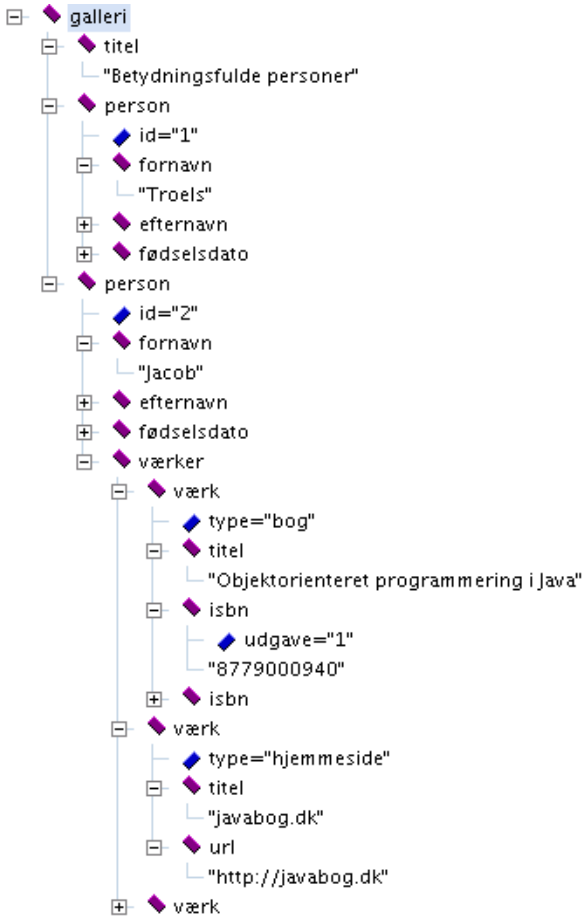
Denne egenskab er en af grundene til at HTML blev en succes: efterhånden som netlæserne udviklede sig og kunne fremvise stadigt mere sofistikerede ting, kunne HTML−koden udvides med flere og flere elementer og attributter og stadigvæk være læsbar og vises rimeligt i gamle netlæsere, der ikke forstod de nye elementer og attributter.

XML har gennemgået en rivende udvikling og der er et så stort antal måder at arbejde med XML på, at det kan virke uoverskueligt til at begynde med.

11.1.2 DOM − Dokumentets objektmodel

DOM (Document Object Model) er en måde at repræsentere et XML−dokument: Hvert element repræsenteres som et objekt med nogle attributter og underelementer, sådan at dokumentet fremstår som *et træ af objekter*.

Udviklingsværktøjer kan nemt vise og navigere i DOM−træet. Her ses DOM−træet af persongalleri.xml som Borland JBuilder viser det (nogle grene i træet er foldet sammen):



Her er 'roden' i træet <galleri>, som har 3 'grene' i form af underelementerne <titel>, <person> og <person>, som hver kan have flere underelementer.

11.1.3 XPath − stier i XML−dokumenter

XML er så smart indrettet, at man kan angive 'stier' til at finde frem til elementer i et givent dokument. Der er et helt sprog, kaldet XPath, som bruges til at udvælge data i DOM−træet.

Disse stier kan være relative, kan vælge flere elementer og en lang række andre ting.

Her er et par eksempler på XPath−udtryk:

- **"galleri/person[@id=2]/fornavn/text()"** anvendt på ovenstående XML−fil persongalleri.xml vil det finde <galleri>−indgangen, under denne finde <person>−indgangen med attributten id=2 og under den finde <fornavn>−indgangens tekst, d.v.s. strengen:

```
Jacob
```

- **"//*/titel/text()"** leder hele træet igennem og finder alle <titel>−koder, uanset hvor det er i DOM−træet:

```
Betydningsfulde personer
Objektorienteret programmering i Java
javabog.dk
Videregående programmering i Java
```

- **"//*/isbn/../titel/text()"** leder træet efter <isbn>−koder, går et niveau op (til <værk>) og finder titlen af værket, sådan at man får en liste over alle værker med ISBN:

```
Objektorienteret programmering i Java
Videregående programmering i Java
```

- **"galleri/person/værker/værk[@type="hjemmeside"]/titel/text()"** finder alle værker af type hjemmeside og udskriver titlen:

```
javabog.dk
```

Ønsker du at vide mere om XPath−udtryk og lave mere avancerede udtryk så se f.eks.:

- 'How XPath Works' i 'The J2EE 1.4 Tutorial': http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JAXPXSLT3.html
- http://www.w3schools.com/xpath/

11.1.4 Transformering af XML−data (XSL)

XML Style Sheet Language (XSL) er en måde at transformere et XML−dokuments data til et hvilken som helst andet tekstformat, f.eks. HTML. Et XSL−dokumenter gør brug af XPath−sproget til at bestemme hvilke dele af XML−dokumentet, der skal ind hvor.

XSL fungerer, populært sagt, lige som god gammeldags brevfletning: XSL−dokumentet er brevfletningsskabelonen, XML−dokumentet er adresselisten. Når de flettes får man breve ud, der ligner skabelonen, men hvor alle felterne i skabelonen er erstattet med data fra XML−dokumentet.

Ønsker du at vide mere om at skrive XSL−transformering så se f.eks.:

- 'Transforming XML Data with XSLT' i 'The J2EE 1.4 Tutorial': http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JAXPXSLT6.html
- http://www.w3schools.com/xsl/

11.1.5 XML−behandling (SAX og DOM)

Der findes to fundamentalt forskellige måder at læse en XML−fil på:

- SAX (Simple Api for XML processing): Filen gennemløbes fra en ende til en anden, og hvert element, der mødes afstedkommer en *hændelse*, der skal fanges.
- DOM (Document Object Model): Hele filen indlæses og hvert element repræsenteres som et objekt med nogle attributter og underelementer, sådan at dokumentet fremstår som et træ af objekter.

Et program der bruger SAX registrerer en lytter, der får kaldt metoder, mens dokumentet gennemløbes. SAX−måden med seriel læsning er hurtig, har lavt hukommelsesforbrug og er velegnet til at trække nogle data ud at et XML−dokument.

DOM−måden er langsommere og bruger mere hukommelse (da hele dokumentet er repræsenteret i hukommelsen), men kan være mere overskuelig at programmere, da den tillader at man hopper frem og tilbage i objekttræet. Den husker alle elementerne, også dem programmet ikke forstår, og er derfor velegnet til at redigere i XML−dokumenter.

11.1.6 Dokumenttypedefinitionen

Et XML−dokument kan (men skal ikke) have tilknyttet en beskrivelse af hvilke elementer, der er tilladt i dokumentet. Der findes to måder at definere dokumentformatet:

- DTD (Document Type Definition) er en ældre standard.
- XML Schema er en nyere standard. XML Schema−dokumenter er i sig selv XML.

DTD er det letteste at læse for mennesker, men XML Schema er lige så stille i gang med at erstatte DTD, bl.a. fordi det kan udtrykke meget mere komplekse regler for dokumenttypens elementer.

Som udvikler kommer man nok aldrig til at læse en dokumenttypedefinition, da der næsten altid er en vejledning henvendt til mennesker, som er meget lettere at læse.

Derfor har de fleste ikke brug for andet end en overfladisk viden om, hvad det er, der skal stå øverst i deres XML−fil, hvis de vil henvise til en dokumenttypedefinition af den ene eller den anden slags.

Eksempel

Webapplikationers driftsbeskrivelsesfil web.xml (se afsnit 7.5) er et eksempel på en XML−fil med en dokumenttypedefinition.

I starten af web.xml−fil i version 2.3 henvises til et DTD:

starten af web.xml i version 2.3 henviser til et DTD

```
<?xml version="1.0"?>
<!DOCTYPE web−app PUBLIC "−//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web−a
<web−app>
```

I web.xml i version 2.4 skiftede man over til XML Schema:

starten af web.xml i version 2.4 henviser til et XML Schema

```
<?xml version="1.0"?>
<web−app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web−app_2_4.xsd"
    version="2.4">
```

Er du virkelig interesseret i at se hvordan dokumenttypedefinitioner ser ud kan du se:

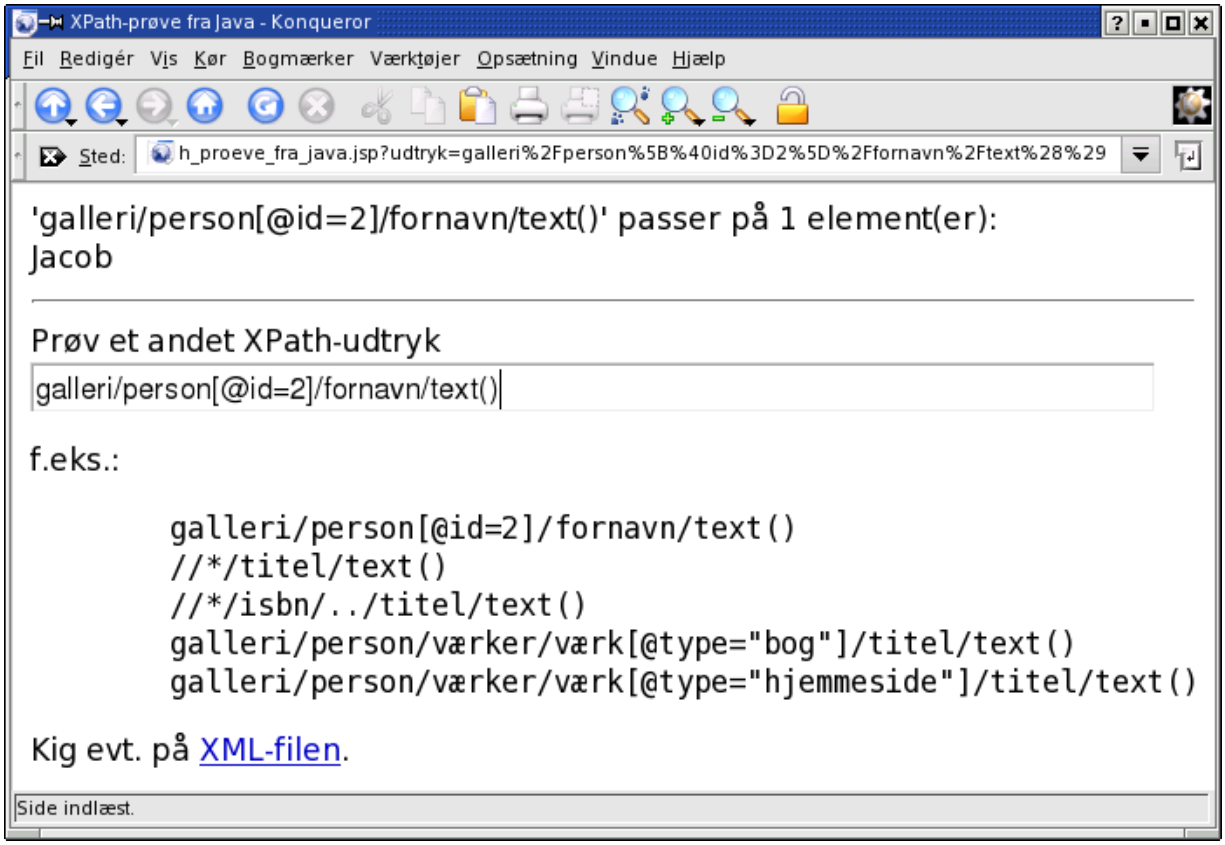
- Eksempel på DTD på http://java.sun.com/dtd/web−app_2_3.dtd
- Eksempel på XML Schema på http://java.sun.com/xml/ns/j2ee/web−app_2_4.xsd

11.2 XML−behandling i Java

Der findes meget stor understøttelse for XML−behandling i Java. Det følgende er derfor kun et par eksempler på, hvordan XML−værktøjerne kan bruges i forbindelse med webprogrammering og JSP.

11.2.1 Brug af DOM og XPath

Her er en JSP−side, der benytter XML−funktioner indbygget i Javas standardbibliotek til at indlæse persongalleri.xml og lade brugeren afprøve forskellige XPath−udtryk til at trække information ud af filen:




```
<%@page import="org.w3c.dom.*,org.apache.xpath.*,javax.xml.parsers.*" %>
<html>
<head><title>XPath−prøve fra Java</title></head>
<body>

<%
  String udtryk = request.getParameter("udtryk");
  if (udtryk == null) udtryk = "galleri/person[@id=2]/fornavn/text()";

  try {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    // find filen persongalleri.xml i samme mappe som denne JSP−side
    String side = application.getRealPath(request.getServletPath());
    String fil = side.substring(0, side.lastIndexOf('/')) + "/persongalleri.xml";
    // Fortolk XML−koden til et DOM−træ
    Document træ = factory.newDocumentBuilder().parse("file:"+fil);

    // Lav liste af resultater fra XPath−udtrykket
    NodeList res = XPathAPI.selectNodeList(træ, udtryk);

    %>'<%= udtryk %>' passer på <%= res.getLength() %> element(er):<br><%
    // Gennemløb listen og udskriv
    for (int i = 0; i < res.getLength(); i++) {
      out.print(res.item(i).getNodeValue()+"<br>");
    }

  } catch (Exception e) {
    e.printStackTrace();
    out.print("<p>Der opstod et problem: "+e+"</p>");
  }
%>

<hr>
Prøv et andet XPath−udtryk
<form><input name="udtryk" value='<%= udtryk %>' type="text" size="70"></form>
f.eks.:
<pre>
  galleri/person[@id=2]/fornavn/text()
  //*/titel/text()
  //*/isbn/../titel/text()
```

```
  galleri/person/værker/værk[@type="bog"]/titel/text()
  galleri/person/værker/værk[@type="hjemmeside"]/titel/text()
</pre>
Kig evt. på <a href="persongalleri.xml">XML-filen</a>.
</body>
</html>
```

Ovenstående eksempel bruger XML-funktioner, der findes i JDK1.4. Bruger du en anden version af JDK kan det være at klassen XPathAPI er ukendt. Det skal du så rette op på ved at hente Xalan-Java fra http://xml.apache.org/xalan-j/.

11.2.2 Nem generering af XML fra Java-objekter

Javabønner (og andre objekter med get- og set-metoder) og almindelige objekter fra Javas standardbibliotek (såsom strenge, ArrayList, Date, ...) kan meget nemt gemmes som XML.

I eksemplet Kalender.java i afsnit 9.4.5 gemmer vi en ArrayList af strenge som XML:

```
  // gem som XML
  XMLEncoder kal = new XMLEncoder(new FileOutputStream("kalender.xml"));
  kal.writeObject(liste);
  kal.close();
```

Lige så nemt som det er at gemme objekter som XML er det at indlæse dem igen:

```
  // indlæs kalenderen fra XML-fil på disken
  XMLDecoder kal = new XMLDecoder(new FileInputStream("kalender.xml"));
  liste = (ArrayList) kal.readObject();
  kal.close();
```

Kigger man i filen kalender.xml på harddisken ser man at den indeholder

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.4.2_03" class="java.beans.XMLDecoder">
 <object class="java.util.ArrayList">
  <void method="add">
   <string></string>
  </void>
  <void method="add">
   <string>Undervise</string>
  </void>
  <void method="add">
   <string></string>
  </void>
  <void method="add">
   <string></string>
  </void>
  <void method="add">
   <string>Holde fri</string>
  </void>
 </object>
</java>
```

Bemærk at klasserne XMLEncoder og XMLDecoder ikke kan anvendes til at konvertere hvad som helst til og fra XML. Specielt skal man være opmærksom, hvis man ønsker at gemme sine egne klasser: De skal være javabønner og alle data i objektet, man ønsker gemt, skal være egenskaber (se afsnit 9.2.2, Egenskaber på en javabønne).

Serialisering

Ønsker man at gemme alle slags objekter er *serialisering* med ObjectOutputStream og ObjectInputStream, der f.eks. er beskrevet på http://javabog.dk en bedre løsning.

```
  // gem som serialiseret objekt (ikke XML)
  ObjectOutputStream kal = new ObjectOutputStream(
                           new FileOutputStream("kalender.ser"));
  kal.writeObject(liste);
  kal.close();
```

og de-serialisering:

```
  kal = new ObjectInputStream(new FileInputStream("kalender.ser"));
  liste = (ArrayList) kal.readObject();
  kal.close();
```

Ulempen med serialisering er, at filen med de gemte objekter (kalender.ser) er i et binært format, der meget vanskeligt kan læses på anden måde end ved at deserialisere det i Java.

11.3 Syndikering (nyhedsfødning)

Indholdssyndikering (eng.: content syndication) går kort sagt ud på at hente information (f.eks. nyheder) fra andre hjemmesider og vise det på sin egen.

Syndikering kan give liv i en hjemmeside og anvendes rigtig meget på næsten alle internetportaler, når de på forsiden viser f.eks. vejrudsigter og nyheder og anden information, som de ikke selv producerer, men henter fra andre steder på nettet.

Syndikering forkortes RSS, der står for (afhængigt af hvem man spørger :−) Really Simple Syndication, Rich Site Summary eller RDF Site Summary. Teknologien udsprang af RDF (Resource Description Framework), et system Netscape oprindelig introducerede.

11.3.1 Nyhedskildernes format

Informationen fra den anden hjemmeside skal være i et bestemt XML−format. For eksempel Java−nyheder fra Sun:

eksempel fra http://servlet.java.sun.com/syndication/rss_java_highlights.xml på RSS−fil med nyheder

```
<?xml version="1.0"?>
<rss version="0.91">
  <channel>
    <title>java.sun.com</title>
    <link>http://java.sun.com/</link>
    <description>java.sun.com is the premier source of information about the Java platform.</description>
    <language>en−us</language>
    <copyright>Copyright: (C) 1995−2003 Sun Microsystems, Inc.</copyright>

    <image>
      <title>java.sun.com</title>
      <url>http://java.sun.com/images/v4_java_logo.gif</url>
      <link>http://java.sun.com/</link>
      <width>38</width>
      <height>66</height>
      <description>Visit java.sun.com</description>
    </image>

    <item>
      <title>Play Ball!</title>
      <link>http://servlet.java.sun.com/logRedirect/rss_java_highlights/http://developer.java.sun.com/developer/
      <description>Tendu's Java software applications give Major League baseball teams instant insight −− and ma
    </item>

    <item>
      <title>Toward a Global "Internet of Things"</title>
      <link>http://servlet.java.sun.com/logRedirect/rss_java_highlights/http://developer.java.sun.com/developer/
      <description>Everything you buy, consume, and enjoy−−even trees!−−may soon be tagged with a unique ID. Lea
    </item>
  ... (flere <item>−indgange)
  </channel>
</rss>
```

Her kunne man f.eks. bruge XPath−udtrykket 'rss/channel/description/text()', for at få beskrivelsen af nyhedskilden (der er: 'java.sun.com is the premier source of information about the Java platform.').

Ovenstående er RSS version 0.91. Det præcise format har gennem tiden ændret sig lidt, så på nettet kan man finde nyhedskilder i RSS−format version 0.91, 0.92, 1.0 og 2.0. De enkelte versioner adskiller sig ikke meget fra hinanden, men DOM−træerne kan være lidt forskellige, så man skal lige tjekke om nyhederne kommer korrekt over på ens egen side.

Her er et eksempel på en RSS−fil version 1.0 med nyheder fra Danmark Radio:

eksempel fra http://www.dr.dk/nyheder/html/nyheder/rss/ på RSS−fil version 1.0 (nogle af indgange er fjernet)

```
<?xml version="1.0" encoding="iso−8859−1"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22−rdf−syntax−ns#"
         xmlns="http://purl.org/rss/1.0/"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:taxo="http://purl.org/rss/1.0/modules/taxonomy/"
         xmlns:syn="http://purl.org/rss/1.0/modules/syndication/">

<channel rdf:about="http://www.dr.dk/nyheder/">
<title>DR Nyheder Online</title>
<link>http://www.dr.dk/nyheder/</link>
<description>Velkommen til DR Nyheder på nettet: Nyheder opdateret 24 timer i døgnet. © DR 2003</description>
<dc:language>da</dc:language>
<items><rdf:Seq>
 <rdf:li rdf:resource="http://www.dr.dk/nyheder/indland/article.jhtml?articleID=194577" />
 <rdf:li rdf:resource="http://www.dr.dk/nyheder/indland/article.jhtml?articleID=194567" />
 <rdf:li rdf:resource="http://www.dr.dk/nyheder/udland/article.jhtml?articleID=194574" />
 <rdf:li rdf:resource="http://www.dr.dk/nyheder/indland/article.jhtml?articleID=194569" />
 <rdf:li rdf:resource="http://www.dr.dk/nyheder/indland/article.jhtml?articleID=194553" />
</rdf:Seq></items>

<item rdf:about="http://www.dr.dk/nyheder/indland/article.jhtml?articleID=194577">
 <title>Læger truer med sygehusstrejke</title>
 <link>http://www.dr.dk/nyheder/indland/article.jhtml?articleID=194577</link>
 <description>En alvorlig konflikt truer med at lamme dele af det vestsjællandske sundhedsvæsen, efter at 70 led
</item>

<item rdf:about="http://www.dr.dk/nyheder/indland/article.jhtml?articleID=194567">
 <title>Sverige − et smuthul for tvangsægteskaber</title>
 <link>http://www.dr.dk/nyheder/indland/article.jhtml?articleID=194567</link>
 <description>Hver måned flytter mellem 48 og 64 unge nydanskere til Sverige for at blive familiesammenført med
</item>

<item rdf:about="http://www.dr.dk/nyheder/udland/article.jhtml?articleID=194574">
 <title>To franske journalister gidsler i Irak</title>
```

```
 <link>http://www.dr.dk/nyheder/udland/article.jhtml?articleID=194574</link>
 <description>Irakiske bortførere har givet Frankrig 48 timer til at gå ind på deres krav, hvis de skal skåne to
</item>

<item rdf:about="http://www.dr.dk/nyheder/indland/article.jhtml?articleID=194569">
 <title>Advarsel mod sød yoghurtdrik</title>
 <link>http://www.dr.dk/nyheder/indland/article.jhtml?articleID=194569</link>
 <description>Ernæringseksperter advarer mod Arlas søde yoghurtdrik Cultura, som de kalder for slik forklædt som
</item>

</rdf:RDF>
```

Sammenligner man ovenstående med eksemplet med Java−nyheder, ser man at DOM−træet er lidt anderledes. Man skal f.eks. bruge 'RDF/channel/description/text()' som XPath−udtryk (med RDF i stedet for rss) for at få beskrivelsen ('Nyheder Danmarks Radio')[1].

11.3.2 Fremvisning med RSS−taglib

Lad os nu se på, hvordan man kan indflette andres nyheder i sine egne sider.

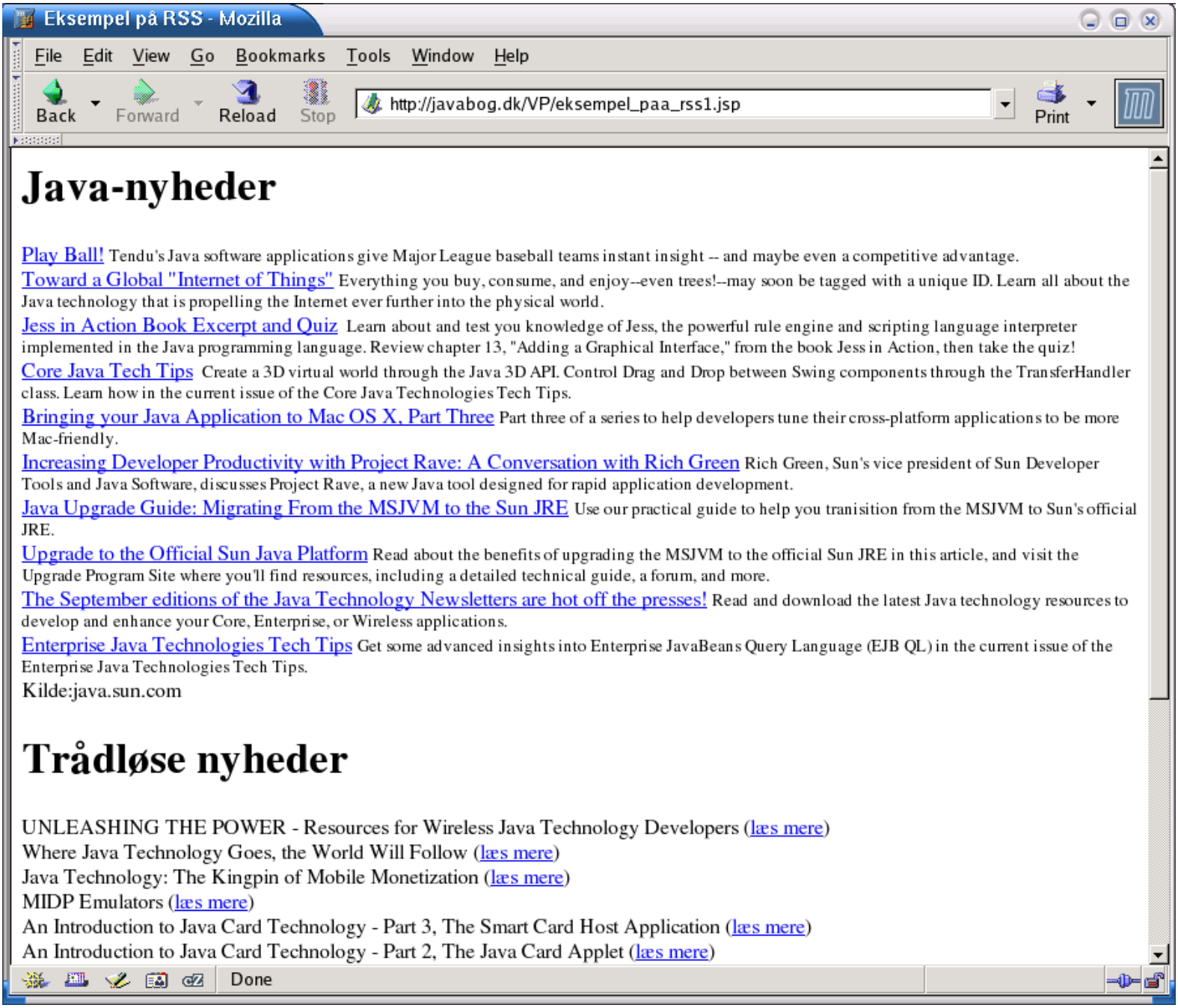
Hvis man ikke har mod på selv at skrive koden, der henter og behandler XML−filerne kan man på http://java.sun.com/developer/technicalArticles/javaserverpages/rss_utilities/ få et JSP taglib−bibliotek der tillader RSS−fødning (af kilder med RSS version 0.91, 0.92 og 2.0) til en webside. Et tag library (forkortet taglib) er, som navnet antyder, et bibliotek af HTML−lignende koder. Ligesom JSP−koderne udføres disse på serveren, sådan at klienten aldrig ser dem.

På hjemmesiden er også beskrevet, hvordan taglib−biblioteket installeres. Dette er en rimelig nem procedure: Hent ZIP−filen, læg rssutils.jar ind i WEB−INF/lib/ i webapplikationen og rssutils.tld ind i WEB−INF/.

Herunder er et simpelt eksempel på brug af RSS−taglibbet:

```
<%@ taglib uri="/WEB-INF/rssutils.tld" prefix="rss" %>
<html>
<head><title>Syndikering med RSS-taglib</title></head>
<body>

<h1>Java-nyheder</h1>
<rss:feed feedId="nyh"
          url="http://servlet.java.sun.com/syndication/rss_java_highlights.xml" />

<rss:forEachItem feedId="nyh">
  <a href='<rss:itemLink feedId="nyh"/>'><rss:itemTitle feedId="nyh"/></a> 
  <font size="-1">
    <rss:itemDescription feedId="nyh"/>
  </font><br>
</rss:forEachItem>
Kilde:<rss:channelTitle feedId="nyh"/><br>

<h1>Trådløse nyheder</h1>
<rss:feed feedId="traadl"
      url="http://servlet.java.sun.com/syndication/rss_wireless_highlights.xml" />
<rss:forEachItem feedId="traadl">
  <rss:itemTitle feedId="traadl"/>
   (<a href='<rss:itemLink feedId="traadl"/>'>læs mere</a>)<br>
</rss:forEachItem>

</body>
</html>
```

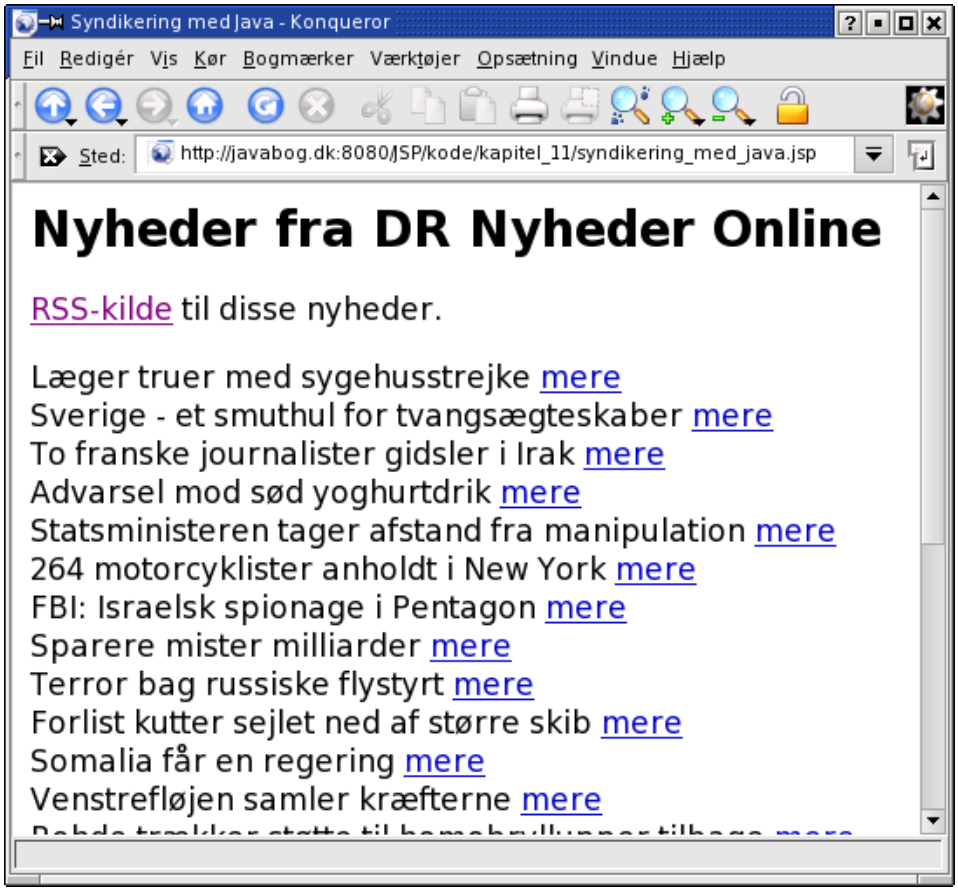
Denne side producerer følgende skærmbillede, med nyheder fra to nyhedskilder (Java−nyheder og nyheder om trådløse apparater) præsenteret lidt forskelligt neden under:



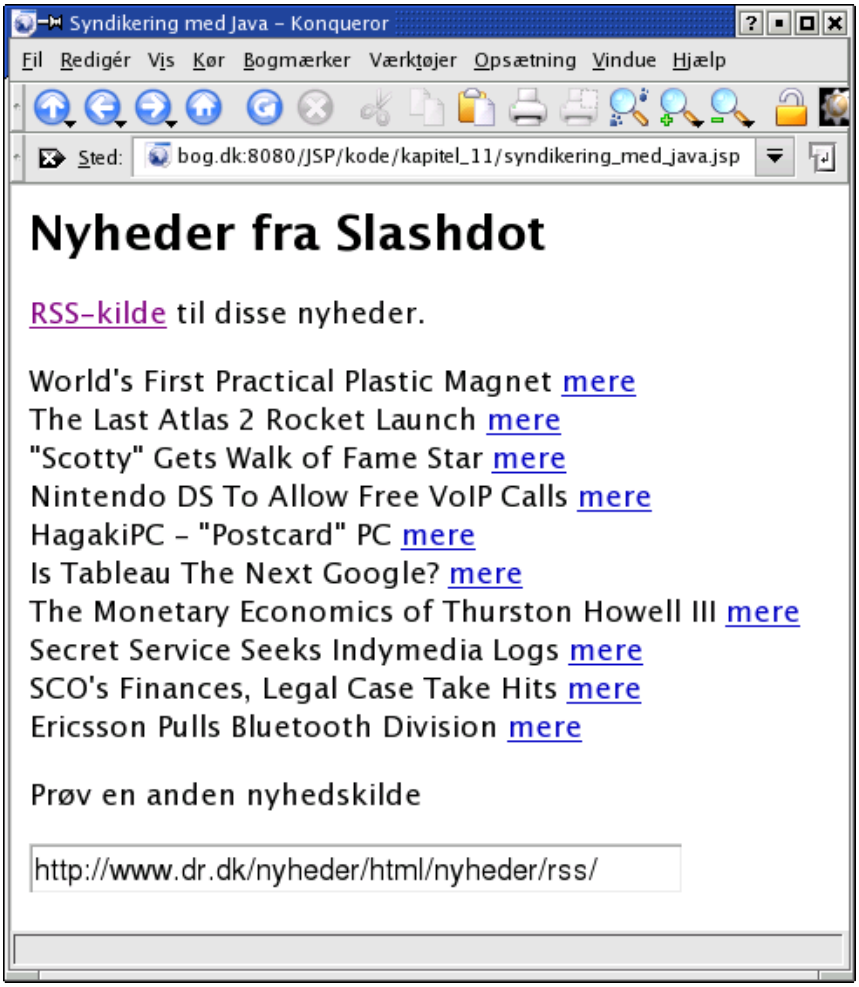
11.3.3 Fremvisning med Java

Har du mod på selv at arbejde med XML, kan du benytte Javas indbyggede funktioner til at udvælge informationer fra en RSS−nyhedkilde med XPath og flette dem ind i dine sider.

Det følgende eksempel fletter nyheder fra Danmarks Radio ind i siden:




```
<%@page import="org.w3c.dom.*,org.apache.xpath.*,javax.xml.parsers.*" %>
<html>
<head><title>Syndikering med Java</title></head>
<body>

<%
  String kilde = request.getParameter("kilde");
  if (kilde == null) kilde = "http://www.dr.dk/nyheder/html/nyheder/rss/";

  try {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    // Fortolk kildens XML-kode til et DOM-træ
    Document træ = factory.newDocumentBuilder().parse(kilde);

    // Find titlen på nyhedskilden med XPath-udtryk
    Node titel = XPathAPI.selectSingleNode(træ, "RDF/channel/title/text()");
  %>

<h1>Nyheder fra <%= titel.getNodeValue() %></h1>
<a href="<%= kilde %>" type="application/rss+xml">RSS-kilde</a> til disse nyheder.
<p>
  <%
    // Lav liste med overskrifter på artikler med XPath-udtryk
    NodeList overskrifter = XPathAPI.selectNodeList(træ, "RDF/item/title/text()");

    // Lav liste over henvisninger til mere læsninger med XPath-udtryk
    NodeList henvisninger = XPathAPI.selectNodeList(træ, "RDF/item/link/text()");

    // Gennemløb listerne og udskriv dem
    for (int i = 0; i < overskrifter.getLength(); i++) {
      out.print(overskrifter.item(i).getNodeValue());
      out.print(" <a href="+henvisninger.item(i).getNodeValue()+">mere</a><br>");
    }
```

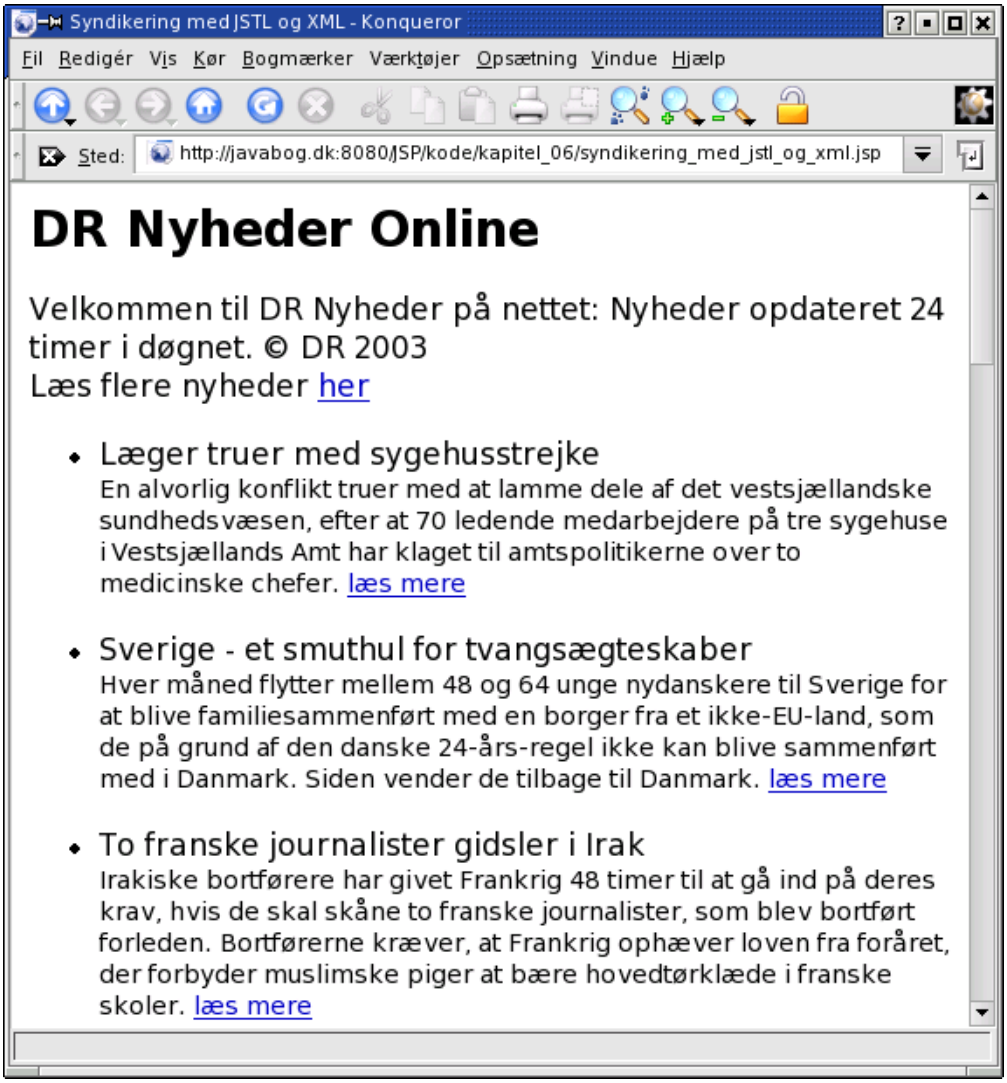



```
  } catch (Exception e) {
    e.printStackTrace();
    out.print("Et problem opstod: "+e);
  }
%>


<p>
Prøv en anden nyhedskilde

<form>
<input type="text" name="kilde"
 value="http://slashdot.org/index.rss"
 size="40">
</form>

</body>
</html>
```

Eksemplet tillader at man kan ændre nyhedskilde. På billedet til højre hentes nyhederne i stedet fra Slashdot (der leverer nyheder for nørder).

11.3.4 Syndikering med JSTL

Man kan også udnytte JSTL til syndikering. I afsnit 6.4.1 vises et eksempel, der indfletter nyheder fra Danmarks Radio ved hjælp af JSTLs XML−funktioner:



I afsnit 6.4.2 er også nævnt lidt om caching af nyhedskilder, som er relevant at overveje, hvis man for alvor vil gøre brug af syndikering.

XPath−udtrykkene i afsnit 6.4.1 er desværre lidt knudrede:

```
<x:out select="$rss//*[name()='channel']/*[name()='title'][1]"/>
```

Det skyldes at JSTLs XML−behandling endnu ikke fuldt understøtter brug af namespaces, som RSS version 1.0 benytter (det er de mange xmlns:−koder i toppen af DRs RSS−fil).

Det kunne egentlig være skrevet så enkelt som (RSS 1.0):

```
<x:out select="$rss/RDF/channel/title/text()"/>
```

eller (RSS 0.91, hvor DOM−træets rod hedder rss og ikke RDF):

```
<x:out select="$rss/rss/channel/title/text()"/>
```

En fordel ved de knudrede XPath−udtryk er dog, at de faktisk fungerer for alle RSS−versionerne, uanset om hele molevitten nu engang ligger under 'RDF' eller 'rss' i DOM−træet.

11.3.5 Mere information og nyhedskilder på nettet

Se http://www−106.ibm.com/developerworks/java/library/j−jstl0520 for mere information om brug af JSTL til RSS−syndikering.

Using RSS in JSP pages: http://today.java.net/pub/a/today/2003/08/08/rss.html

Der findes utallige nyhedskilder på nettet, og der kommer dagligt flere til.

Ofte vil et websted vise et lille XML−ikon nederst på siden for at gøre opmærksom på at dens indhold kan syndikeres. Det ser sådan her ud: XML.

Her er et par websteder med RSS og URLen til deres nyhedsfødning:

- DR Nyheder: http://www.dr.dk/nyheder/html/nyheder/rss/

- Dagbladet Information http://rss.asdf.dk/information.rss
- Alt Om København: http://rss.asdf.dk/aok.rss
- Børsen Online: http://rss.asdf.dk/borsen.rss
- Slashdot: http://slashdot.org/slashdot.rdf
- Java−nyheder: http://servlet.java.sun.com/syndication/rss_java_highlights.xml
- Trådløs Java: http://servlet.java.sun.com/syndication/rss_wireless_highlights.xml

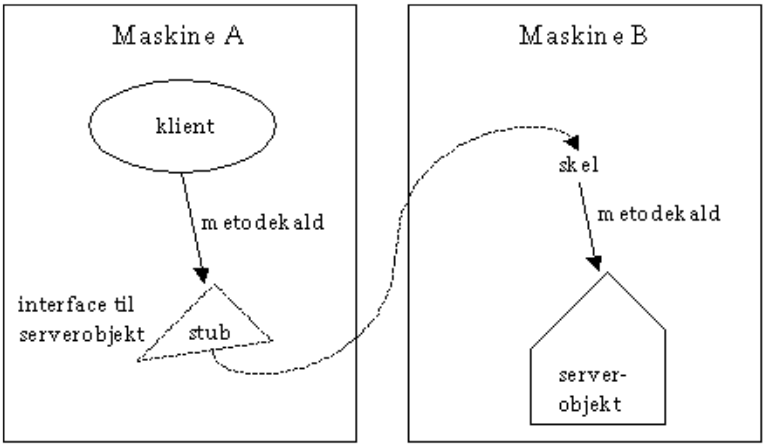
Her er et par steder, der holder styr på de danske websteder med RSS−fødning:

- http://rss.spontek.dk
- http://rss.asdf.dk

11.4 Principper i metodekald over netværk

I dette afsnit vil vi beskrive en måde at arbejde med objekter, der eksisterer på en anden fysisk maskine, *som om de var lokale objekter*.

Herunder er tegnet, hvad der sker, når en klient på maskine A laver et metodekald i et *serverobjekt* (da: værts−objekt), der befinder sig på maskine B:



Serverobjektet findes slet ikke på maskine A, i stedet er der en såkaldt *stub*, et proxy−objekt, der repræsenterer det rigtige objekt på serveren.

Når der sker et kald til stubben på maskine A, sørger den for at transportere kaldet og alle parametre til maskine B, hvor serverobjektet bliver kaldt, som om det var et lokalt kald. Serverobjektets svar bliver transporteret tilbage til stubben, der returnerer svaret til klienten.

Denne proces foregår helt automatisk og er usynlig for klienten såvel som serverobjektet.

For at kunne transportere parametre og returværdi mellem maskinerne, skal alle data, der sendes over netværket, kunne omsættes til en række byte af systemet ('serialiseres').

11.4.1 Systemer til metodekald over netværk

I dag findes der mange måder, hvorpå man kan lave systemer, der kan håndtere program−til−program kommunikation, bl.a. RMI, EJB, webtjenester, CORBA eller DCOM.

Det er ovenstående princip, der bruges i alle disse systemer.

- RMI, Remote Method Invocation er meget nemt at benytte. Det bygger på Java, så både klientdel og serverdel skal skrives i Java. RMI benytter Java−serialisering (der er et binært format), når det sender data mellem klient og server. Man kan ikke forvente, at kunne kalde metoder i et RMI−objekt uden for lokalnettet, da RMI benytter andre porte end HTTP−protokollen (næsten alle firewalls har åbnen på port 80, som HTTP−protokollen benytter sig af).
- CORBA, Common Object Request Broker Architecture, er sproguafhængig (så objekterne behøves ikke skrives i Java), men er svært at bruge, der skal genereres et væld af filer og CORBAs beskrivelse af fjernobjekternes metoder, IDL (Interface Definition Language), er svært at forstå.
- EJB, Enterprise JavaBeans, der er beskrevet i kapitel 12, benytter RMI eller CORBA.
- Webtjenester, beskrevet i næste afsnit, er sprog− og platformsuafhængigt, benytter XML i stedet for binære protokoller og kan benytte HTTP−protokollen til at transportere metodekald (så kommunikationen slipper igennem firewalls).
- DCOM, Distributed Common Object Model, er Microsofts måde at kommunikere fra et program til et andet. Selvom programmeringssprogene kan variere, er det kun programmeringssprog, der er kompatible på en Microsoft−platform, der kan benyttes, såsom Visual Basic, C++ osv. Det samme gælder for klientprogrammerne, der også skal ligge på Microsoft−platforme.

11.5 Webtjenester

Webtjenester (eng.: Web services) så dagens lys første gang i starten af 2002 og er udviklet af Sun Microsystems, da de lancerede deres første udgave af Java Web Services Developer's Pack (JWSDP).

En webtjeneste er et system til program−til−program kommunikation på tværs af netværk. Man kommunikerer med en webtjeneste ved at sende og modtage SOAP−meddelelser (XML−dokumenter, se senere), som skal overholde det format, der er fastlagt i beskrivelsen af webtjenesten (WSDL, se senere). SOAP−meddelelserne overføres typisk med HTTP−protokollen. Webtjenester kan være registreret offentligt (UDDI, se senere).

11.5.1 SOAP − kommunikation med webtjenesten

Kommunikationen mellem en klient og en webtjeneste foregår med SOAP (Simple Object Access Protocol), hvor meddelelserne mellem klient og server er små XML−dokumenter i bestemt format.

SOAP er hverken system− eller programmeringssprogsafhængigt: Både webtjenesterne og de klienter, der bruger dem, kan skrives på alle mulige platforme såsom Linux, Mac eller Windows og med forskellige programmeringssprog såsom Java, C++, Perl, Python, ASP eller PHP. En webtjeneste og de tilhørende klienter behøver ikke at være skrevet i det samme programmeringssprog, så længe kommunikationen mellem dem overholder den struktur, der er beskrevet i webtjenestens WSDL.

11.5.2 WSDL − beskrivelsen af webtjenesten

WSDL står for Web Service Definition Language. Det er et XML−dokument, der beskriver en webtjeneste i detaljer. I WSDLen til en webtjeneste kan man se webtjenestens URL − den adresse, hvor webtjenesten udbydes, hvilke metoder webtjenesten stiller til rådighed, hvilke parametre, som webtjenestens metoder tager i en forespørgsel og hvilke parametre webtjeneste svarer tilbage med på en forespørgsel.

Hvis man vil danne sine egne stubbe til at tilgå en webtjeneste, skal man have fat i webtjenestens WSDL, så man kan se, hvad metoderne hedder, og hvordan de skal forespørges.

Der findes mange programmer, der automatisk kan danne stubbene ud fra et WSDL. Hvordan man gør i Oracle JDeveloper og Apache Axis er omtalt i afsnit 11.5.5, Genere Java−API fra webtjeneste (WSDL).

11.5.3 UDDI − offentlige lister over webtjenester

Skal man udgive sin webtjeneste til offentlig brug, kan det være smart at udgive den i en slags telefonbog, som De Gule Sider. Brugere kan slå op i denne 'telefonbog' for at finde en tjeneste, der dækker deres behov. Her kan de finde URLen på webtjenesten, webtjenestens WSDL og en beskrivelse af, hvad webtjenesten kan gøre for dem. Der er et par af de store IT−firmaer, der stiller en sådan telefonbog til rådighed blandt andet Microsoft og IBM.

F.eks. kan man finde de webtjenester, som internet−boghandlen Amazon stiller til rådighed, på: https://uddi.ibm.com/ubr/findservice?action=details&servicekey=BA6D9D56−EA3F−4263−A95A−EEB17E5910DB.

En sådan telefonbog over webtjenester kaldes for UDDI, som står for Universal Description and Discovery Integration. Fordelen ved at få listet sine webtjenester i UDDIerne er at de bliver kendt et sted, hvor brugerne ved, de skal kigge efter dem.

Øvelse

Gå ind på UDDI−siden http://uddi.xmethods.net/ og læs om de forskellige webtjenester.

På siden er også en grænseflade, som kan bruges til at prøve at kalde nogle af webtjenesterne. Prøv også den.

11.5.4 Bruge Googles webtjenester til søgninger

Google har en webtjeneste, som tillader udviklere at skrive programmer, der kan søge i de milliarder af dokumenter, som Google indekserer.

Gå ind på http://google.com/apis og læs om Google Web APIs og hent udviklingskittet.

Du skal også bruge en nøgle, som du kan få ved at registrere dig. Nøglen er en lang streng som f.eks.:
VkjZ2P5QFHICg5MfvjuwzmJm1T/iYbdY

Det letteste er at bruge Googles Java−programmeringsgrænseflade (API) til webtjenesten.
Pak filen ud og prøv demoen, der ligger i googleapi.jar. Prøv f.eks. fra kommandolinjen at søge på "Den lille havfrue" (skriv på en enkelt linje − husk at udskifte med din egen nøgle):

```
java −cp googleapi.jar com.google.soap.search.GoogleAPIDemo
  VKjZ2P5QFHICg5MfvjuwzmJm1T/iYbdY search "Den lille havfrue"
```

Ud skulle gerne komme en masse adresser på sider om den lille havfrue, f.eks.:

```
Parameters:
Client key = VkjZ2P5QFHICg5MfvjuwzmJm1T/iYbdY
Directive  = search
Args       = Den lille havfrue
Google Search Results:
======================
{
TM = 0.367159
Q  = "Den lille havfrue"
Start Index = 1
End   Index = 10
```

```
Estimated Total Results Number = 7590
Rs =
  {
  [
  URL  = "http://hjem.get2net.dk/OSJ_INDEX/hybenrose/havfruen/"
  Title = "<b>Den</b> <b>lille</b> <b>Havfrue</b>"
  Snippet = "En beretning om Danmarks nationale vartegn, <b>Den</b> <b>lille</b> <b>Havfrue</b> ved Langelinie −
  Summary = "Uofficiel side om <b>Den</b> <b>lille</b> <b>Havfrue</b> ved Langelinie − Fakts om havfruen, eventy
  Cached Size = "10k"
  ],

  [
  URL  = "http://hjem.get2net.dk/chenero/hca/hcaev008_da.html"
  Title = "<b>Den</b> <b>lille</b> <b>havfrue</b>"
  Snippet = "<b>Den</b> <b>lille</b> <b>havfrue</b>. af Hans Christian Andersen. <b>...</b> ?Det gør ondt!? sagd
  Summary = ""
  Cached Size = "47k"
  ],

  [
  URL  = "http://www.hcandersen−homepage.dk/skulptur_den_lille_havfrue.htm"
  Title = "HC Andersen: Skulptur &amp;quot;<b>Den</b> <b>lille</b> <b>havfrue</b>&amp;quot;"
  Snippet = "..... Skulptur &amp;quot;<b>Den</b> <b>lille</b> <b>havfrue</b>&amp;quot;. hans christian andersen homepage
  Cached Size = "14k"
  ],
  }
}
```

11.5.5 Genere Java−API fra webtjeneste (WSDL)

Du kan også genere APIet i dit eget sprog ud fra webtjenestebeskrivelsen (WSDL) med et værktøj. Herunder er beskrevet, hvordan et API kan genereres i Java ud fra en webtjeneste med Oracle JDeveloper og Apache Axis.

Oracle JDeveloper

Åbn beskrivelsen af tjenesten (WSDL−filen), højreklik på den og vælg "Generate Web Service Stub/Skeleton" og generér en klient−side stub med en main()−metode.

Apache Axis

I Apache Axis' API kan man få dannet en WSDL på to måder:

1. Den ene måde er, at idriftsætte sin webtjeneste ud fra en XML−fil, som er en WSDD−fil, WebService Definition Document. Axis vil derefter starte webtjenesten og danne WSDL−filen. Man kan derefter se filen, ved at skrive webtjenestens URL i adressefeltet på sin netlæser, men efter webtjenestens navn tilføje parameteren wsdl, f.eks.: http://localhost:8080/jboss−net/services/Veksletjeneste**?wsdl**
2. Man kan også benytte det medfølgende program, Java2WSDL, som kan producere WSDL−dokumentet ud fra den Java−klasse, som man vil udbyde som en webtjeneste. Programmet Java2WSDL tager 3 parametre, ud over Javaklassens fulde navn: Navnet på WSDL−filen, webtjenestens URL og et "target namespace", som skal være et entydigt navn på webtjenesten.

Øvelse

Gør ovenstående med Googles WSDL−fil (GoogleSearch.wsdl). Kig filerne igennem og gå derpå ind i GoogleSearchServiceStub og ret main() til:

```
  // opret stubben
  GoogleSearchServiceStub stub = new GoogleSearchServiceStub();

  // foretag søgningen
  GoogleSearchResult sr=stub.doGoogleSearch("VKjZ2P5QFHICg5MfvjuwzmJm1T/iYbdY",
    "Den lille havfrue",new Integer(0),new Integer(10),
    Boolean.FALSE,null,Boolean.FALSE,null,null,null);

  // gennemløb resultaterne
  ResultElement[] re = sr.getResultElements();
  for (int i=0; i<re.length; i++)
    System.out.println(re[i].getTitle() +" "+re[i].getURL());
```

... og kør programmet (husk at udskifte nøglen med din egen nøgle). Ud skulle gerne, som før, komme en række adresser på sider om den lille havfrue.

11.5.6 Generere webtjeneste (WSDL) fra Java−klasse

Opret eller find en simpel klasse med nogle metoder, du kunne tænke dig at udbyde som en webtjeneste.

Lav f.eks. en klasse, der kan konvertere mellem dollar og euro:

```
package webtjeneste;

public class Veksler
{
  public double euroTilDollar(double euro)
  {
```

```
    System.out.println("euroTilDollar("+euro+") kaldt!");
    return euro/1.1;
  }

  public double dollarTilEuro(double dollar)
  {
    System.out.println("dollarTilEuro("+dollar+") kaldt!");
    return dollar*1.1;
  }
}
```

Oracle JDeveloper

- Højreklik derefter på klassen og vælg 'Create J2EE Web Service'.
- Dobbeltklik på den genererede fil og giv tjenesten et navn (f.eks. navnet MinTjeneste).
- Højreklik på filen og vælg 'Run'

... og du har en webtjeneste kørende!

Tjenesten kører i Oracle JDevelopers interne webserver. For at afprøve den, kan du genere en Java−klient fra WSDL−webtjenestebeskrivelsen, som beskrevet i forrige afsnit.

Apache Axis

Axis stiller et program til rådighed, der kan danne Javastubbene til kommunikation med en webtjeneste ud fra webtjenestens WSDL. Dette program hedder WSDL2Java og tager to parametre: et pakkenavn til stub−klasserne og filnavnet på WSDL−filen.

11.5.7 Avanceret: Strukturen i WSDL− og SOAP−filer

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

11.5.8 Yderligere læsning

- Suns portal om webtjenester:
  http://java.sun.com/webservices/
- Apache−gruppens implementering af webtjenester (Axis):
  http://ws.apache.org/axis/java/
- Oracle JDeveloper har meget kraftig understøttelse af webtjenester. En lang række artikler om webtjenester og Oracle JDeveloper kan ses på
  http://www.oracle.com/technology/tech/webservices/htdocs/series/ og
  http://www.oracle.com/technology/tech/webservices/
- Søgeportalen Googles webtjenester:
  http://www.google.com/apis/
- E−handelssiden Amazons webtjenester:
  http://soap.amazon.com/
- Auktionssiden Ebays webtjenester:
  http://developer.ebay.com/DevProgram/
- Liste over firmaer, der udbyder webtjenester:
  http://uddi.xmethods.net/

11.6 Test dig selv

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

11.7 Resumé

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

1Vigtigste forskel er nok at <item>−indgangene i RSS 1.0 er *sideordnet* med <channel>, hvor de i version 0.91 ligger *under* <channel>−indgangen.

12 Enterprise JavaBeans

12.1 J2EE−platformen

12.2 EJB−bønner

12.2.1 Kildeteksten i en bønne

12.2.2 Eksempel: Veksler

12.2.3 EJB−Containerens information om Veksler

12.3 Bruge en EJB−bønne

12.3.1 JNDI (Java Naming and Directory Interface)

12.3.2 Brug af Veksler−bønnen fra en klient

12.3.3 Brug af Veksler fra en webapplikation

12.3.4 Avanceret: Lokale interfaces

Jeg lover at anskaffe den i nær fremtid.

12.4 Typer af EJB−bønner

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.
Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.4.1 Sessionsbønner

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.4.2 Entitetsbønner

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.4.3 Meddelelses−drevne bønner

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.4.4 Tilstandsfuld sessionsbønne: Brugeradgang

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.4.5 Guide til sessionsbønner i Oracle JDeveloper

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.5 Entitetsbønner

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.5.1 Eksempel på en entitetsbønne

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.5.2 Fremfindingsmetoder

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.5.3 Idriftsætte bønnen

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.5.4 Bruge bønnen

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.5.5 Fremfindingsmetoder og EJB Query language

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.5.6 EJB 2.0 og EJB Query language (EJBQL)

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.5.7 Opgaver

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.5.8 Entitetsbønne med CMP fra tabel i JDeveloper

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.5.9 Øvelse i EJBQL

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.6 Transaktioner

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.6.1 Transaktioner i JDBC (resumé)

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.6.2 Transaktioner i en EJB−container

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.6.3 Deklarativ transaktionsstyring af metodekald

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.7 Test dig selv

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

12.8 Resumé

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

13 Internationale sider

Dette kapitel forudsætter kapitel 9, Javabønner i JSP−sider.

13.1 Internationale programmer

Når et program skal anvendes af flere kulturer og sprog, opstår behov for, at programtekster, beløb og dato angives i de pågældende landes sprog og man må i gang med at internationalisere og lokalisere programmet.

Internationalisering (eng.: Internationalization, også kaldet I18N) består i, at gøre programmet sprogneutralt, ved at sørge for, at al formatering og fortolkning af tal−, beløbs−, dato− og tidsangivelser, sker afhængigt af sproget og at al sproglig tekst er flyttet til separate filer. Det er typisk programmørens, der står for denne opgave.

Lokalisering (eng.: Localization, også kaldet L10N) består i, at oversætte filerne fra hovedsproget til de andre sprog, applikationen skal fungere på. Denne opgave varetages typisk af en professionel oversætter.

13.1.1 Formatering af tidspunkter

DateFormat formaterer Date−objekter til strenge (og den anden vej).

```
import java.text.*;
import java.util.*;
public class BenytDateFormat
{
  public static void main(String arg[])
  {
    DateFormat klformat, datoformat, dkf;
    klformat   = DateFormat.getTimeInstance(DateFormat.MEDIUM);
    datoformat = DateFormat.getDateInstance(DateFormat.FULL);
    dkf = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,DateFormat.SHORT);

    Date tid = new Date();
    System.out.println( tid );
    System.out.println( "Kl   :"+ klformat.format(tid) );
    System.out.println( "Dato :"+ datoformat.format(tid) );
    System.out.println( "Tid  :"+ dkf.format(tid) );
  }
}
```

```
Wed Feb 05 14:23:46 GMT+00:00 2003
Kl   :14:23:46
Dato :5. februar 2003
Tid  :05−02−2003 14:23
```

Læg for det første mærke til, at toString() på Date ikke er lokaliseret. Den bør kun bruges til test−udskrifter og logning og ikke i tekst, som brugeren skal læse.

Ovenstående program er kørt med danske sprogindstillinger. Med amerikanske sprogindstillinger bliver uddata i stedet:

```
Mon Dec 03 13:27:57 GMT+01:00 2001
Kl:   1:27:57 PM
Dato: Monday, December 3, 2001
Tid:  Dec 3, 2001 1:27 PM
```

Det ses, at det afhænger en del af sproget, præcist hvordan tider bliver formateret (f.eks. er ugedag med i den amerikanske tekst, men ikke i den danske).

Ønsker man som programmør fuld kontrol over, hvordan teksten bliver formateret, må man selv specificere formatet med SimpleDateFormat:

```
import java.text.*;
import java.util.*;
public class BenytSimpleDateFormat
{
  public static void main(String arg[])
  {
    DateFormat df = new SimpleDateFormat("EEEE 'den' d. MMMM 'år' yyyy.");

    Date tid = new Date();
    System.out.println( df.format(tid) );
  }
}
```

```
mandag den 3. december år 2001.
```

Dermed bliver selve formaterings−strengen sprogspecifik (og den bør lægges ud i en resursefil − se senere). Køres den med amerikanske sprogindstillinger, bliver uddata:

```
Monday den 3. December år 2001.
```

13.1.2 Formatering af tal og beløb

På samme måde som med tidsangivelser formateres/fortolkes tal ved at bede om et formateringsobjekt, der klarer netop denne form for tal:

```
import java.text.*;
public class BenytNumberFormat
{
  public static void main(String arg[])
  {
    double tal = 1234.5678;
    NumberFormat fmt1 = NumberFormat.getInstance();
    NumberFormat fmt2 = NumberFormat.getCurrencyInstance();
    NumberFormat fmt3 = NumberFormat.getPercentInstance();
    System.out.println( fmt1.format(tal) );
    System.out.println( fmt2.format(tal) );
    System.out.println( fmt3.format(tal) );
  }
}
```

```
1.234,568
kr 1.234,57
123.457%
```

Med amerikanske sprogindstillinger bliver uddata:

```
1,234.568
$1,234.57
123,457%
```

13.1.3 Bruge Locale−objekter

Land og sprog (lokalindstillingerne) repræsenteres ved et Locale−objekt. Dette objekt bruges til al formatering, til at finde resursebundter frem og lignende.

Ønsker man kontrol over hvilket sprog, der anvendes, kan man selv oprette et Locale−objekt, og det kan så bruges til at fremskaffe formateringsobjekter til det pågældende sprog:

```
import java.text.*;
import java.util.*;
public class BenytDateFormat2
{
  public static void main(String arg[]) {
    DateFormat klformat, datoformat;

    Locale fransk = new Locale("fr","FR");
    klformat   = DateFormat.getTimeInstance(DateFormat.SHORT,fransk);
    datoformat = DateFormat.getDateInstance(DateFormat.LONG, fransk);

    Date tid = new Date();
    System.out.println( "Kl:   "+ klformat.format(tid) );
```

```
    System.out.println( "Dato: "+ datoformat.format(tid) );
  }
}
```

```
Kl:   14:23
Dato: 5 février 2003
```

13.1.4 De mulige lokalindstillinger

Man kan undersøge hvilke Locale−objekter, der er tilgængelige, ved at kalde klassemetoden Locale.getAvailableLocales(). Her er et lille program, der skriver dem ud:

```
import java.util.*;
public class MuligeSprog
{
  public static void main(String arg[]) {
    Locale[] lokalindstillinger = Locale.getAvailableLocales();
    for (int i=0; i<lokalindstillinger.length; i++) {
      Locale lokalindst = lokalindstillinger[i];
      System.out.print(lokalindst+" ");
      // udkommentér hvis du vil se hvad sproget hedder på sproget selv
      //System.out.println(lokalindst.getDisplayName(lokalindst));
    }
    System.out.println( );
  }
}
```

```
ar ar_AE ar_BH ar_DZ ar_EG ar_IQ ar_JO ar_KW ar_LB ar_LY ar_MA ar_OM ar_QA ar_SA ar_SD ar_SY ar_TN ar_YE hi_IN iw
zh_CN zh_HK zh_TW be be_BY bg bg_BG ca ca_ES cs cs_CZ da da_DK de de_AT de_CH
de_DE de_LU el el_GR en_AU en_CA en_GB en_IE en_IN en_NZ en_ZA es es_AR es_BO
es_CL es_CO es_CR es_DO es_EC es_ES es_GT es_HN es_MX es_NI es_PA es_PE es_PR
es_PY es_SV es_UY es_VE et et_EE fi fi_FI fr fr_BE fr_CA fr_CH fr_FR fr_LU hr
hr_HR hu hu_HU is is_IS it it_CH it_IT lt lt_LT lv lv_LV mk mk_MK nl nl_BE nl_NL
no no_NO no_NO_NY pl pl_PL pt pt_BR pt_PT ro ro_RO ru ru_RU sh sh_YU sk sk_SK sl
sl_SI sq sq_AL sr sr_YU sv sv_SE tr tr_TR uk uk_UA en en_US eo
```

Locale−objektet består af tre dele:

- Første del er sprogkoden, f.eks. da, sv, no, en, fr.
- En valgfri anden del er landekoden, f.eks. DK, NO, GB, DE, FR
- En valgfri tredje del er varianten inden for sprogområdet (f.eks. om valutaen er i euro).

Eksempler:

fr: Fransk
fr_BE: Fransk i Belgien
fr_CA: Fransk i Canada
no_NO: Norsk i Norge (bokmål)
no_NO_NY: Nynorsk i Norge (variant NY er nynorsk)

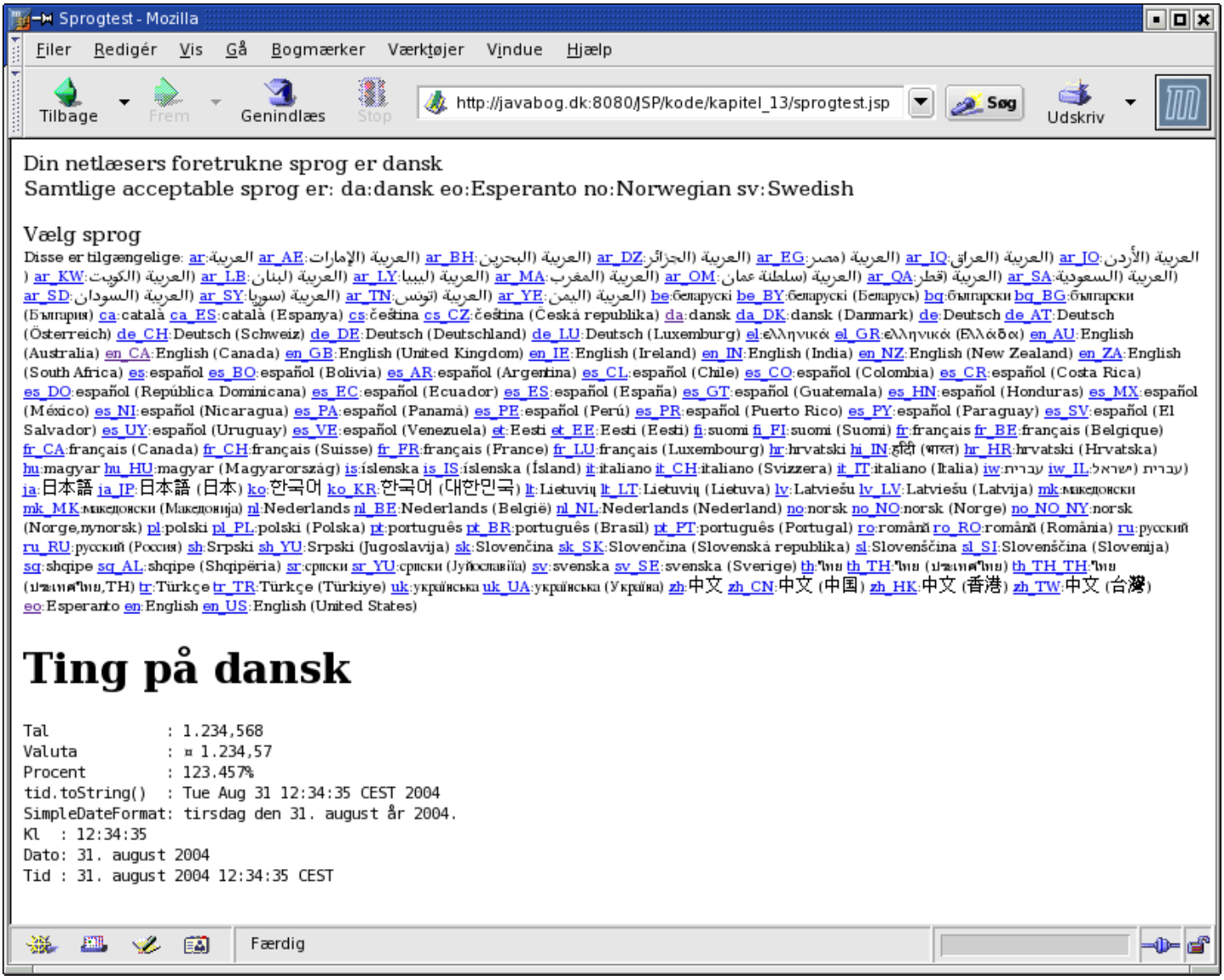
13.2 Eksempel: Webside med alle sprog

Her er et eksempel, der sammenfatter det ovenstående og hvor man kan vælge mellem alle de mulige sprog og se hvordan forskellige ting skrives på dette sprog:

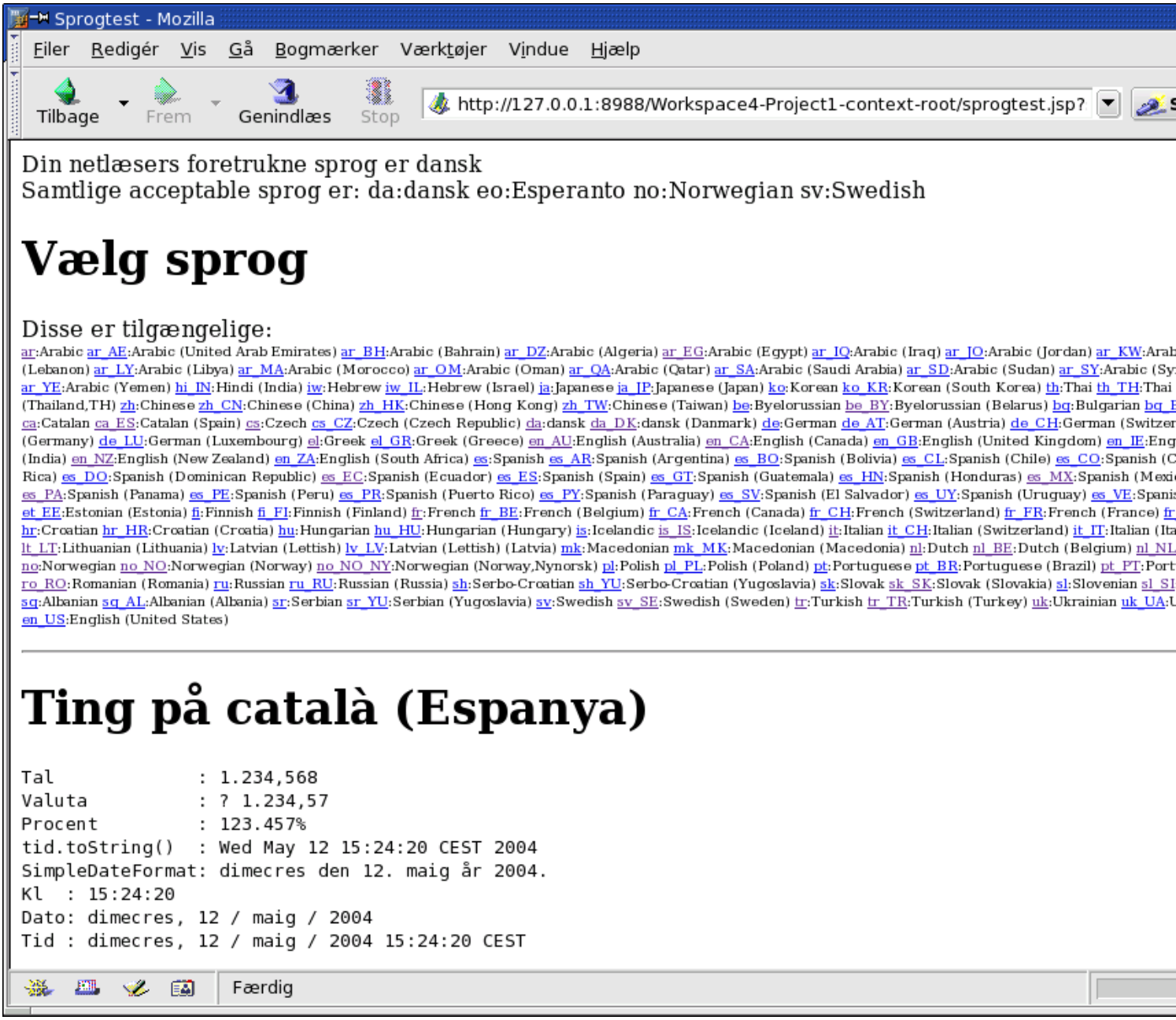
sprogtest.jsp

```
<%@ page import="java.util.*,java.text.*" contentType="text/html;charset=UTF−8" %>
<html>
<head><title>Sprogtest</title></head>
<body>

<p>Din netl&aelig;sers foretrukne sprog er <%=request.getLocale().getDisplayName()%>
<br>Samtlige acceptable sprog er:
<%
  Enumeration e = request.getLocales();
  while (e.hasMoreElements()) {
    Locale l = (Locale) e.nextElement();
    out.print(l + ":"+l.getDisplayName()+"\n" );
  }
%>
</p>

<p>V&aelig;lg sprog<br>
<font size="−1">Disse er tilg&aelig;ngelige:
<%
  Locale[] lokalindstillinger = Locale.getAvailableLocales();
  for (int i=0; i<lokalindstillinger.length; i++) {
    Locale lokalindst = lokalindstillinger[i];
    out.print(" <a href='sprogtest.jsp?sprog="+lokalindst+"'>"
      +lokalindst+"</a>:"                    // sprogkode, f.eks.: da_DK
      +lokalindst.getDisplayName(lokalindst));// fuldt navn paa sproget selv
  }
%>
</font></p>
```

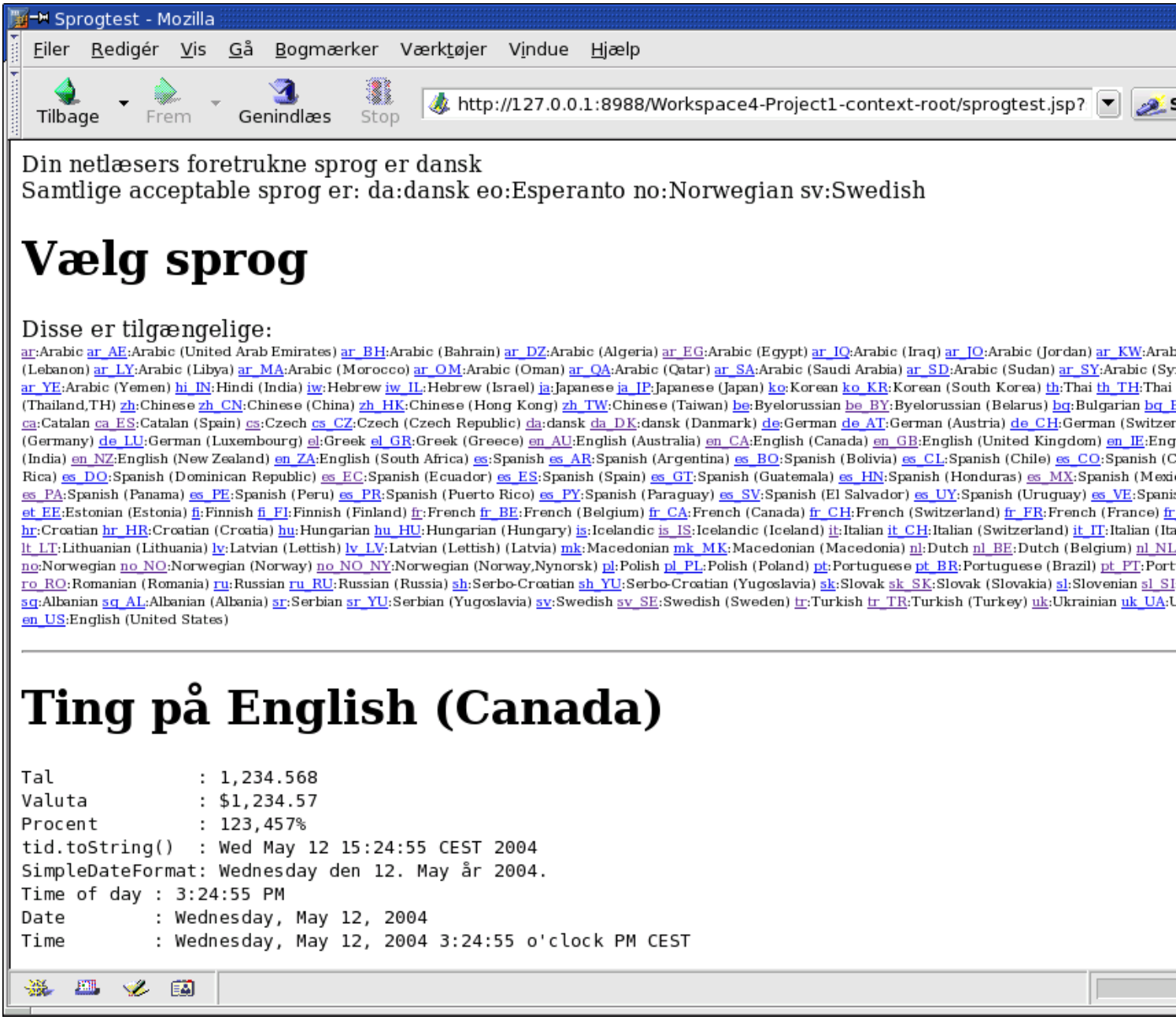
```
<%
  Locale sprog = request.getLocale();             // netlaeserens foretrukne sprog
  String vsprog = request.getParameter("sprog");  // har bruger valgt et andet?
  if (vsprog != null) {
    String[] s = vsprog.split("_");               // split f.eks. da_DK op
    if (s.length==1) sprog = new Locale(vsprog);        // sprog (da)
    else if (s.length==2) sprog = new Locale(s[0],s[1]);// sprog, land (da_DK)
    else sprog = new Locale(s[0],s[1],s[2]);            // sprog, land, variant
  }
%>

<h1>Ting p&aring; <%= sprog.getDisplayName(sprog) %></h1>
<pre>
<%
  double tal = 1234.5678;
  NumberFormat fmt1 = NumberFormat.getInstance(sprog);
  NumberFormat fmt2 = NumberFormat.getCurrencyInstance(sprog);
  NumberFormat fmt3 = NumberFormat.getPercentInstance(sprog);
  out.println( "Tal             : " + fmt1.format(tal) );
  out.println( "Valuta          : " + fmt2.format(tal) );
  out.println( "Procent         : " + fmt3.format(tal) );

  Date tid = new Date();
  out.println( "tid.toString()  : "+ tid );

  SimpleDateFormat brugerdefineretFormat =
              new SimpleDateFormat("EEEE 'den' d. MMMM '&aring;r' yyyy.", sprog);
  out.println( "SimpleDateFormat: " + brugerdefineretFormat.format(tid) );

  DateFormat klformat, datoformat, dkf;
  klformat   = DateFormat.getTimeInstance(DateFormat.MEDIUM, sprog);
  datoformat = DateFormat.getDateInstance(DateFormat.FULL, sprog);
  dkf = DateFormat.getDateTimeInstance(DateFormat.FULL, DateFormat.FULL, sprog);

  ResourceBundle res = ResourceBundle.getBundle("sprogtest.Tekster", sprog);

  out.println( res.getString("Kl_")+ klformat.format(tid) );
  out.println( res.getString("Dato_")+ datoformat.format(tid) );
  out.println( res.getString("Tid_")+ dkf.format(tid) );
%></pre>
</body>
</html>
```



Vælger man nu catalansk i Spanien (ca_ES) får man:



Vælger man engelsk i Canada (en_CA) får man:



Læg mærke til, at de nederste tre tekster (Kl, Dato og Tid) bliver vist på engelsk (Time of day, Date og Time). Det skyldes, at disse tekster er blevet lagt ud i en separat fil (en resursefil), der er blevet oversat til engelsk (se senere).

13.3 Tegnsæt og HTML−entiteter

Læg mærke til linjen

```
<%@ page contentType="text/html;charset=UTF-8" %>
```

øverst i eksemplet.

Sidedirektivet her specificerer at HTML−siden kommer i indkodningen UTF−8, d.v.s. Unicode. Unicode er nødvendigt, for at kunne repræsentere tegn uden for intervallet 0−256, men er ikke kompatibelt ved vestlig indkodning (ISO−8859−1) uden for ASCII−området (de amerikanske tegn A−Z), så æ, ø og å og andre specielle tegn går tabt.

Når man skriver sine websider, må man derfor enten sørge for at siderne er er indkodet i Unicode−formatet eller benytte HTML−entiteter, såsom æ (for æ), ø (for ø) og å (for å) i HTML−koden i stedet.

Det har vi gjort i eksemplet, f.eks har vi skrevet:

```
Din netl&aelig;ser
```

i stedet for:

```
Din netlæser
```

13.4 Tekstindhold i resursefiler

Programmet BenytDateFormat er halvt internationaliseret, da tidsformateringen korrekt skifter afhængigt af sproget. Der mangler kun strengene "Kl: ", "Dato: " og "Tid: ".

Lad os nu fuldføre internationaliseringen ved at lægge tekstindholdet (strengene) ud i et resursebundt (eng.: resource bundle) med navnet Tekster.

Disse kan tilgås fra programmet således:

```
package sprogtest;

import java.text.*;
import java.util.*;
public class BenytDateFormatMedResurser
{
  public static void main(String arg[])
  {
    ResourceBundle res = ResourceBundle.getBundle("sprogtest.Tekster");
    DateFormat klformat, datoformat, dkf;
    klformat   = DateFormat.getTimeInstance(DateFormat.MEDIUM);
    datoformat = DateFormat.getDateInstance(DateFormat.FULL);
    dkf = DateFormat.getDateTimeInstance(DateFormat.MEDIUM,DateFormat.SHORT);

    Date tid = new Date();
    System.out.println( res.getString("Kl_")+ klformat.format(tid) );
    System.out.println( res.getString("Dato_")+ datoformat.format(tid) );
    System.out.println( res.getString("Tid_")+ dkf.format(tid) );
  }
}
```

```
Kl  : 14:23:50
Dato: 5. februar 2003
Tid : 05-02-2003 14:23
```

Resursebundtet, der skal ligge i filen sprogtest.Tekster.properties, indeholder teksten i nøgle−værdi−par.

Disse vil blive brugt, hvis der ikke findes resursefiler for det pågældende sprog:

```
Tid_=Tid  \:
Kl_=Kl   \:
Dato_=Dato \:
```

Køres programmet (på et styresystem indstillet til dansk), fås samme udskrift som før.

Lad os nu lokalisere programmet til engelsk.

Det består i, at vi opretter sprogtest.Tekster**_en**.properties med de engelske tekster:

```
Tid_=Time       \:
Kl_=Time of day\:
Dato_=Date       \:
```

Starter vi derefter programmet med engelske sprogindstillinger, får vi uddata:

```
Time of day : 3:07:28 PM
Date        : Monday, December 3, 2001
Time        : Dec 3, 2001 3:07 PM
```

13.4.1 Hvordan der søges efter resurser

Når programmet skal finde en programtekst ud fra sprogindstillingerne, sker det ved først at kigge i den mest specifikke resursefil. Hvis programteksten ikke findes der, kigges i en mere generel resursefil og til sidst i den mest generelle.

For eksempel vil resurser til sproget fr_CA (canadisk fransk) blive søgt

1. først i Tekster_fr_CA.properties
2. dernæst i Tekster_fr.properties
3. og sidst i Tekster.properties

13.4.2 Avanceret: Binære resursefiler

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

13.4.3 Avanceret tekstformatering

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

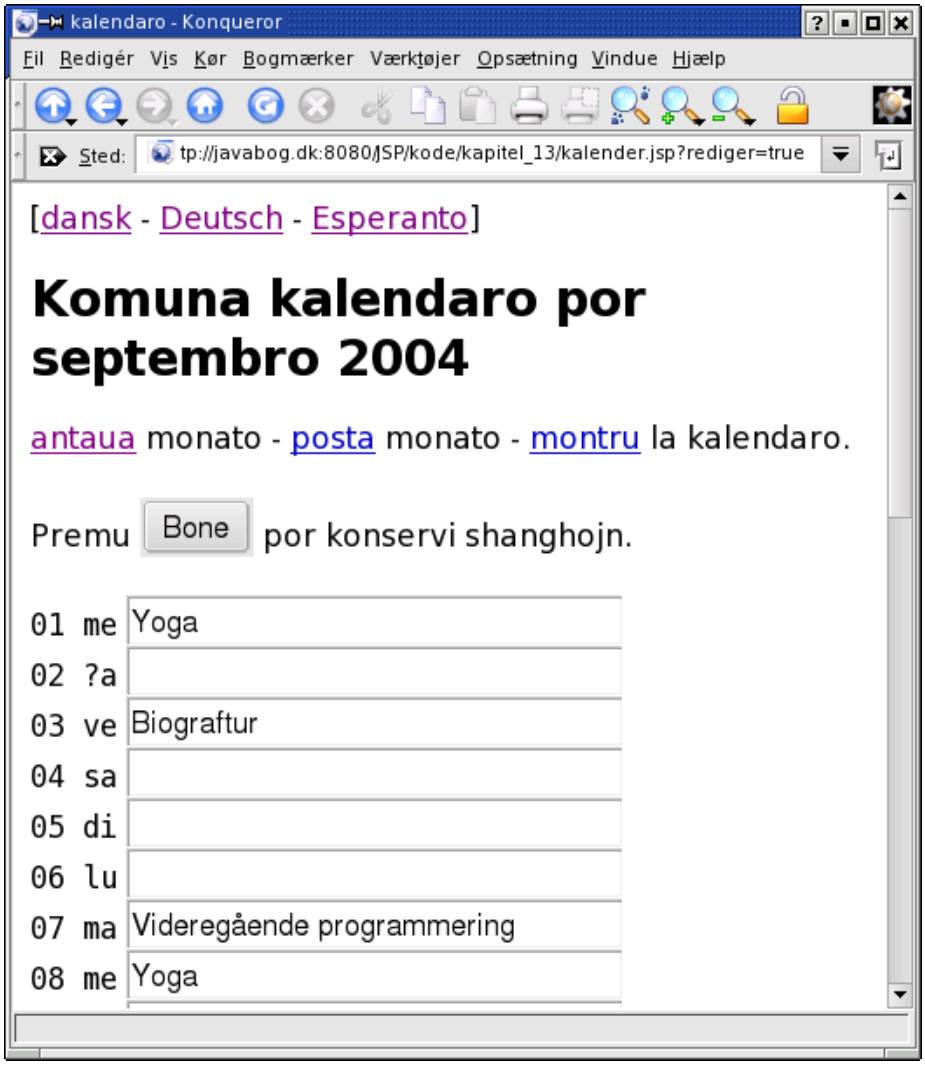
13.5 En international fælleskalender

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

13.5.1 Slutbrugerens oplevelse

Her ses kalenderen på dansk og tysk:

Her er vist redigering på sproget esperanto:



13.5.2 Fuld separation mellem layout og sprog

Eksemplet gør brug af resursebundter til samtlige programtekster og opnår derved, at JSP−siden kun beskæftiger sig med layout og visningslogik.

Oversættelseskoden (opslag i resursebundter) er lagt i en separat fil, nemlig i en Sprogdata−bønne, sådan at JSP−siden kan holdes overskuelig og sådan, at den er så lidt forstyrret af opslag i resursebundter som muligt (hver gang en tekstbid skal ind i JSP−siden kalder den metoden ovs() på bønnen med nøglen).

Her er kalendertekster.properties, med originalsproget (dansk):

```
# Filnavn: kalendertekster.properties
Fælleskalender_for=Fælleskalender for
kalenderen=kalenderen
kalender=kalender
måned=måned
forrige=forrige
næste=næste
redigér=redigér
vis=vis
Tryk_input_type_submit_value_OK_når_du_vil_gemme_ændringer=Tryk <input type='submit' value='OK'> når du vil gemm
_input_type_submit_value_OK_=<input type='submit' value='OK'>
```

Her er kalendertekster_de.properties (tysk):

```
# Filnavn: kalendertekster_de.properties
Fælleskalender_for=Der Gruppenkalender für
kalenderen=der Kalender
kalender=Kalender
måned=monat
forrige=vorheriger
næste=nächste
redigér=redigieren
vis=zeigen
Tryk_input_type_submit_value_OK_når_du_vil_gemme_ændringer=Drücken <input type='submit' value='OK'> um die Änder
_input_type_submit_value_OK_=<input type='submit' value='OK'>
```

Her er kalendertekster_eo.properties (esperanto):

```
# Filnavn: kalendertekster_eo.properties
```

```
Fælleskalender_for=Komuna kalendaro por
kalenderen=la kalendaro
kalender=kalendaro
måned=monato
forrige=antaua
næste=posta
redigér=redaktu
vis=montru
Tryk_input_type_submit_value_OK_når_du_vil_gemme_ændringer=Premu <input type='submit' value='Bone'> por konservi
_input_type_submit_value_OK_=<input type='submit' value='Bone'>
```

Her er javabønnen Sprogdata, der holder styr på sprog og resursebundter:

Sprogdata.java

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

public class Sprogdata
{
  private Locale sprog = Locale.getDefault();
  private ResourceBundle tekster;
  private Bruger bruger;

  public void setLocale(Locale sproget)
  {
    sprog = sproget;
    tekster = ResourceBundle.getBundle("kalendertekster",sprog);
    bruger.setLocale(sprog);
  }

  public void setBruger(Bruger brugerobjekt)
  {
    bruger = brugerobjekt;
    bruger.setLocale(sprog);
  }

  public void setSprog(String sproget) {
    String[] s = sproget.split("_");
    if (s.length==1) setLocale(new Locale(sproget));
    else if (s.length==2) setLocale(new Locale(s[0],s[1]));
    else setLocale(new Locale(s[0],s[1],s[2]));
  }

  /** Oversæt en tekst (slå op i det relevante resursebundt) */
  public String ovs(String nøgle)
  {
    try {
      return tekster.getString(nøgle);
    } catch (MissingResourceException e)
    {
      System.err.println("Advarsel: Manglende resurse i kalendertekster:\n"
          + nøgle.replaceAll("[\\p{Punct}\\s]+","_")  // erstat andre tegn med _
          + "=" + nøgle.replaceAll("[\r\n]","\\\n") );
      return nøgle;
    }
  }
}
```

Her er JSP−siden med kalenderen:

kalender.jsp

```
<%-- Knyt bønnen Bruger til brugerens session under navnet b --%>
<jsp:useBean id="b" class="kalender.Bruger" scope="session" />
 <%-- <title>kalender</title> --%>
<%-- Overfør parametre til b, der svarer til egenskaber --%>
<jsp:setProperty name="b" property="*" />

<%-- Knyt bønnen Sprogdata til brugerens session under navnet s --%>
<jsp:useBean id="s" class="kalender.Sprogdata" scope="session">
  <% // Første gang: Knyt til Bruger−objekt og registrér sproget
    s.setBruger( b );
    s.setLocale( request.getLocale() ); %>
</jsp:useBean>

<%-- Overfør evt. sprog−parameter til s --%>
<jsp:setProperty name="s" property="sprog" />

<html>      <%-- s.ovs() oversætter teksten til det pågældende sprog --%>
<head><title><%= s.ovs("kalender") %></title></head>
<body>

[<a href='kalender.jsp?sprog=da'>dansk</a> −
 <a href='kalender.jsp?sprog=de'>Deutsch</a> −
 <a href='kalender.jsp?sprog=eo'>Esperanto</a>]

<h1><%= s.ovs("Fælleskalender_for") %> <%= b.getDatostr() %></h1>
```

```
<a href="kalender.jsp?maaned=<%= b.getMaaned()-1 %>"><%= s.ovs("forrige") %></a>
  <%= s.ovs("måned") %> -
<a href="kalender.jsp?maaned=<%= b.getMaaned()+1 %>"><%= s.ovs("næste") %></a>
  <%= s.ovs("måned") %> -

<%
  if (b.isRediger()) { // redigering - vis en input-formular
%>
  <a href="kalender.jsp?rediger=false"><%= s.ovs("vis") %></a>
    <%= s.ovs("kalenderen") %>.<br>
  <br>
  <form action="kalender.jsp" method="post">
    <%=s.ovs("Tryk_input_type_submit_value_OK_når_du_vil_gemme_ændringer")%>.<br>

    <% b.udskrivDagsprogram(out); %>

    <br><br>
    <%= s.ovs("_input_type_submit_value_OK_") %>
  </form>
<%
  } else {             // fremvisning
%>
  <a href="kalender.jsp?rediger=true"><%= s.ovs("redigér") %></a>
    <%= s.ovs("kalenderen") %>.<br>

  <% b.udskrivDagsprogram(out); %>
<%
  }
%>

</body>
</html>
```

Sammenlign med kalender.jsp i afsnit 9.4.3 (ændringer er markeret med fed).

13.6 Yderligere læsning

Om internationalisering af JSP-sider:

http://java.sun.com/webservices/docs/1.3/tutorial/doc/WebI18N.html

Generelt om Java og internationalisering:

http://java.sun.com/docs/books/tutorial/i18n/

Liste over alle sprog- og landekoder kan du se i afsnit 13.1.4 og afsnit 13.2.

Stikordsregister

STYLE="background: transparent">

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

14 Rettelser og tilføjelser

I dette kapitel finder du rettelser og tilføjelser til den trykte bog (rettelserne er ført ind på http://javabog.dk). Du kan udskrive kapitlet og gemme bagest i din bog.

14.1 Vigtige rettelser til 1. udgave

I afsnit 3.3, Appendiks: Typer af formularfelter, ret (tak til Klaus Elmquist Nielsen):

```
<input type="hidden">
```

Skjult felt. Vises for brugeren, men alligevel har en værdi, se afsnit 3.6.2, Skjulte felter i formularer.

Til:

```
<input type="hidden">
```

Skjult felt. Vises *ikke for brugeren, men har alligevel* en værdi, se afsnit 3.6.2, Skjulte felter i formularer.

I log_ind.jsp i afsnit 9.5.3, Login−siden, ret:

```
<font color="red">
<%
if (login.isLoggetInd()) {
```

Til:

```
<font color="red">
<%
login.tjekLogin();
if (login.isLoggetInd()) {
```

I VekslerBean.java i afsnit 12.2.2, Eksempel: Veksler, ret:

```
    System.out.println("dollarTilEuro("+dollar+euro+") kaldt!");
```

Til:

```
    System.out.println("dollarTilEuro("+dollar+") kaldt!");
```

En række steder i kapitel 11 og kapitel 12 skal linjen

```
  public double euroTilDollar(int euro)
```

i Veksler−klassen rettes til

```
  public double euroTilDollar(double euro)
```

side 227, afsnit 11.7, punkt 1 i listen:

XML kræver at alle attributter skal være i "..", hvoraf

<motor hk=100 />

skal være

<motor hk=″100″ />

14.2 Større tilføjelser til 1. udgave

Afsnit 3.7.6 Sende mere data til klienten løbende og de tre efterfølgende afsnit.

Afsnit 5.8.6 Metadata og afsnit 5.8.7 Eksempel: Webgrænseflade til database.

Kapitel 15 om Java Server Faces.

14.2.1 Sende mere data til klienten løbende

Nogen gange kan en anmodning tage lang tid at fuldføre. Det kan f.eks. være en krævende operation eller et indviklet databaseopslag.

I sådanne tilfælde vil brugeren opleve at siden tager utrolig lang tid at indlæse. Da er det hensigtsmæssigt at sende *noget* af svaret på anmodningen, f.eks. titel og overskrift på siden, gerne nogle instruktioner eller forbehold for hvordan det efterfølgende resultat (når det kommer) skal læses.

I andre tilfælde kommer der løbende data som klienten skal se efterhånden som de dukker op.

Man tvinger serveren tømme sine interne buffere og sende al data til klienten ved at kalde

```
  out.flush();
```

Derved sendes HTTP−hovedet[1] og det, der indtil videre er skrevet til out−objektet.

14.2.2 Eksempel: Syvtabellen langsomt

Her er eksemplet Syvtabellen fra afsnit 2.2.1 skrevet sådan at der går et sekund mellem hver linje:

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

<%
  for (int i=1; i<=10; i++)
  {
    out.flush();        // tøm interne buffere, sådan at data sendes til klienten
    Thread.sleep(1000); // vent et sekund
%>
    Syv gange <%= i %> er: <%= 7*i %>.<br>
<%
  }
%>
</p>
</body>
</html>
```

14.2.3 Eksempel: Følge med i serverens logfil

Det følgende eksempel følger med i Tomcats logfil (logs/catalina.out) og opdaterer hvert sekund siden med de nyeste oplysninger fra logfilen.

```
<html>
<head><title>Serverlog</title></head>
<body>
<h1>Webserverens logfil</h1>
Det følgende er det nyeste fra webserverens logfil.<br>
Hvert sekund opdateres siden hvis der kommer mere i loggen, i de næste 10 minutter.
<pre>
<%
  String filnavn = "logs/catalina.out"; // sti til Tomcats logfil
  long glFilLgd = 0;                    // hvor mange byte filen fylder
  byte[] buf = null;                    // en databuffer
  java.io.RandomAccessFile fil = null;  // ... og selve filen

  for (int i=0; i<600; i++)             // gennemløb 60 sekunder * 10 minutter
  try {
    Thread.sleep(1000); // vent et sekund

    fil = new java.io.RandomAccessFile(filnavn,"r");   // åbn filen

    if (glFilLgd < fil.length())                       // har længden ændret sig?
    {
      if (glFilLgd==0) glFilLgd = fil.length()−1500;  // læs sidste 1500 tegn
      int bufLgd = (int) (fil.length() − glFilLgd);
      if (buf==null || bufLgd > buf.length) buf = new byte[bufLgd];
      fil.seek(glFilLgd);
      fil.read(buf,0,bufLgd);
      String tekst = new String(buf,0,bufLgd);
      out.println("−−−− lgd="+fil.length()+" tid="+new java.util.Date()+" −−−−");
      out.write(tekst);
      out.flush();     // tøm interne buffere, sådan at data sendes til klienten
      glFilLgd = fil.length();
    }
  } finally {
    fil.close();
  }
%>
</pre>
Slut. <br>
Vil du se mere af logfilen må du genindlæse siden.
</body>
</html>
```

Bemærk at det kan være en sikkerhedsrisiko at kunne følge med serverloggen (her kan ubetænksomme programmører f.eks. komme til at logge brugernavne og adgangskoder som andre helst ikke skal have adgang til).

14.2.4 Metadata

xxx tekst ind

14.2.5 Eksempel: Webgrænseflade til database

Under udviklingen af en webapplikation, der bruger en database, kan det være rart med en webbaseret grænseflade til databasen som man kan skrive SQL−kommandoer ned i eller lave forespørgsler med.

```
<%@ page language="java" import="java.sql.*" %>
<html>
<head><title>Webgrænseflade til database</title></head>
<body>

<h1>SQL−grænseflade til databasen</h1>
Herunder kan du skrive SQL−kommandoer. Adskil kommandoer med ;
<form method="post">
<textarea name="sql" rows="8" cols="70">SELECT * FROM kunder
</textarea>
<input type="submit" name="udf" value="Udfør SQL!">
</form>

<%
  // Se om der kommer nogle SQL−kommander der skal behandles
  String sql = request.getParameter("sql");
  Connection con = null;
  Statement stmt = null;
  if (sql != null) {
    Class.forName("com.mysql.jdbc.Driver");
    con = DriverManager.getConnection("jdbc:mysql:///test");
    stmt = con.createStatement();
    String[] kommandoer = sql.split(";"); // opdel i kommandoer efter skilletegn ;
    for (int k=0; k<kommandoer.length; k++) try {
      String kom = kommandoer[k].trim();
      if (kom.length()==0) continue; // videre til næste linje
      out.println("<hr>Udfører '"+kom+"':<br>");
      if (kom.toUpperCase().startsWith("SELECT")) {
        ResultSet rs = stmt.executeQuery(kom);
        ResultSetMetaData rsmd = rs.getMetaData();
        int antalKolonner = rsmd.getColumnCount();

        out.println("<table border='2'><tr>");
        for (int i=1; i<=antalKolonner; i++)
          out.println("<th>"+rsmd.getColumnName(i)+"</th>");
        out.println("</tr>");

        // udskriv cellerne i hver række
        while (rs.next())
        {
          out.println("<tr>");
          for (int i=1; i<=antalKolonner; i++)
            out.println("<td>"+rs.getString(i)+"</td>");
          out.println("</tr>");
        }
        out.println("</table>");
        rs.close();
      } else {
        int ret = stmt.executeUpdate(kom);
        out.println("Returkode: "+ret);
      }
    } catch (Exception e) {
      out.println("Der skete en fejl: "+e);
      e.printStackTrace();
    }
    stmt.close();
    con.close();
  }
%>
</body>
</html>
```

Tillægskapitel til Webprogrammering med JSP

Jacob Nordfalk

http://javabog.dk/

30−03−05

1Og derved kan man så ikke mere foretage omdirigeringer, j.v.f. afsnit 3.7 og afsnit 4.3.

 javabog.dk | << forrige | indhold | næste >> | programeksempler | om bogen

15 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

15.1 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

15.2 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

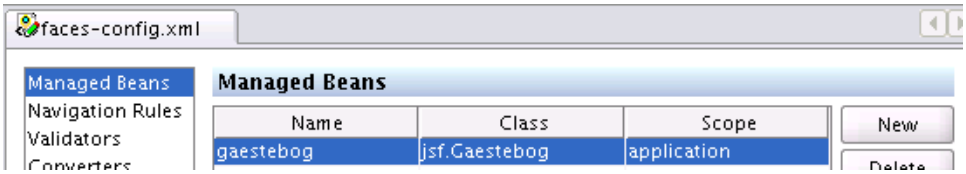
15.3 Visning af gæstebog fra JSF

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>
```

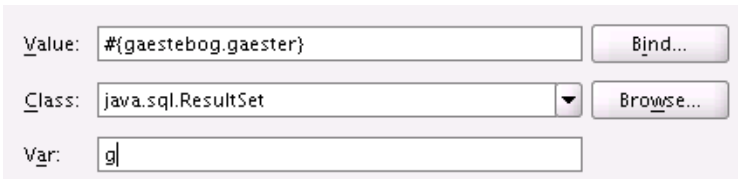
I JDeveloper (og sikkert også andre værktøjer) kan det gøres ved at gå hen på fanen 'Overview' på faces-config.xml:



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'.

15.4 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.

15.4.1 Avanceret: Validatorer

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

15.4.2 Avanceret: Fejlmeddelelser

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

15.4.3 Avanceret: Resursebundter

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

15.4.4 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 15.6.1At 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.

15.4.5 Avanceret: Startværdier for egenskaber

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

15.5 Aflæsning af JSF−bønnes egenskab

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

15.5.1 Adgang til de implicitte objekter fra JSF

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 'omgivelser' eller på engelsk 'external context'), nemlig request−objektet attributter (gaest−bønnen er har virkefelt 'request'). Dem kan vi få med

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

Adgang til request−objektet fra JSF

Ville vi have fat i hele request−objektet skulle vi skrive

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

Fordi JSF i princippet kan fungere oven på andre systemer end Java Server Pages er returværdien af getRequest()−metoden af type blot 'Object' og vi må selv konvertere den til et HttpServletRequest−objekt.

Tilsvarende gælder for de andre implicitte objekter defineret implicit i JSP.

Adgang til session−objektet fra JSF

Man kan få adgang til session−objektet med:

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

Kaldet til getSession(false) giver sessionobjektet hvis det allerede er oprettet, ellers null. Ønsker man at sessionobjektet skal oprettes hvis det ikke allerede findes skal man skrive:

```
HttpSession session = (HttpSession) fc.getExternalContext().getSession(true);
```

Adgang til application−objektet fra JSF

Man får fat i application−objektet med

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

```
  ServletContext application;
  application = (ServletContext) fc.getExternalContext().getContext();
```

Bemærk at

```
  fc.getApplication()
```

ikke giver det normale application−objekt man er vant til! xxx

15.6 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.

15.6.1 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 15.10) 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;
    }
  }
}
```

15.7 Resumé

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

15.7.1 JSF og JDeveloper

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

15.7.2 Måder at aktivere JSF−komponenter

Dette afsnit er ikke omfattet af Åben Dokumentslicens.
Du skal købe bogen for at måtte læse dette afsnit. Jeg erklærer, at jeg allerede har købt bogen
Jeg lover at anskaffe den i nær fremtid.

15.8 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

15.8.1 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'

15.8.2 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.

15.8.3 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

15.9 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).

15.10 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 nøgle−objekt. 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[2].

```
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"));
    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
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.

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

2Bruger 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.
