



HØGSKOLEN I SØR-TRØNDELAG
Avdeling for informatikk og e-læring - AITeL

Løsningsforslag

Kandidatnr:	
Eksamensdato:	3. desember 2009
Varighet:	0900-1100
Emnekode:	LO348D/LN349D
Emnenavn:	Web-applikasjoner med JSP og JSF
Klasse(r):	2 ING, nettstudenter
Studiepoeng:	
Faglærer(e):	Else Lervik, Grethe Sandstrak
Kontaktperson (adm.)	
Hjelpemidler:	Ingen
Oppgavesettet består av:	2 oppgaver og 6 sider (inkludert forside og vedlegg)
Vedlegg består av:	2 sider
Merknad: Oppgaveteksten kan beholdes av studenter som sitter eksamenstiden ut.	
Lykke til!	

Oppgave 1 (JSP og JSF) – vekt 25%

Et nettsted kan ha mange besøkende og utføre mange ulike interaksjoner med de ulike brukerne. Når du programmerer en nettside kan det være nødvendig å assosiere data med den enkelte besøkende på nettsiden. Beskriv kort ulike måter vi kan gjøre dette på. Gi også eksempel på typiske bruksområder for de ulike metodene du beskriver. (Se bort fra datafiler og databaser i denne oppgaven.)

Her forventes en redegjøring for følgende metoder.

Omskriving av URL: utvider url med de dataene man ønsker å overføre mellom nettsidene. Eksempel: nettside.no?farge=Red. Sensitive data bør ikke overføres på denne måten da de sendes i klartekst og klient kan i prinsippet endre på dette før det oversendes tjener og videre behandling.

- Sessionid dersom cookies er slått av, personalisering av grensesnitt

Skjulte felt: lager ”hidden” felt som ikke vises når du laster siden som normalt. Feltet vises dersom du ser på koden for fila. (vis –kilde). Ingen sensitive data bør overføres på denne måten da de sendes i klartekst og klient kan i prinsippet endre på dette før det oversendes tjener og videre behandling.

- Sideteller, quiz (huske svar, spørsmålsnr.),

Cookies: Informasjonskapsel, opprettes på tjenersiden og sendes til klienten for permanent lagring der. Obs, klienten kan slå av støtte for cookies og klienten kan slette informasjonskapslene. Hver kapsel har satt en viss levetid (etterpå slettes den). Hver gang en klient besøker et nettsted vil eventuelle cookies tilhørende dette nettstedet oversendes tjener – eventuelle endringer på kapselen utføres og den sendes så tilbake. Informasjonskapsler har begrenset størrelse (4 kb), kan kun inneholde ren tekst, maks 20 informasjonskapsler pr. nettsted og max 300 totalt pr. klient. Ikke lurt å lagre sensitive data som for eksempel passord da dette lagres i klartekst. Levetid til informasjonskapsel kan settes.

- grensesnittvalg, språkvalg, brukerid, personalisering av gui, session_id, annonsører for å tilpasse reklame

Session : informasjon knyttet til en klient via en unik session_id, lagres på tjeneren til sesjonen har vært inaktiv innen max_inaktiv tid, eller klient har lukket nettleser. SesjonsId lagres i en Cookie, dersom klienten har støtte for cookies slått på. Hvis ikke, sendes sesjonsid via omskriving av URL. Ulike datatyper kan lagres i et sesjonsobjekt.

– handlekurv, personalia, huske sist besøkte side

Oppgave 2 (JSP) – vekt 20%

Vi har et vareregister og har laget en jsp-side for å registrere nye varer i registeret. Til å hjelpe oss har vi laget en Java-klasse – *Vare*, som ligger i pakken *EksDes2009*. Skjemaet med tilhørende kode ser slik ut:

<h2>Registrer varer</h2> <p>Varenavn: <input type="text"/></p> <p>Pris: <input type="text"/></p> <p><input type="submit" value="Registrer ny vare"/></p>	<pre> <html> <body> <h2>Registrer varer</h2> Varenavn: <input type="text" name="navn"/> Pris: <input type="text" name="pris"/> <input type="submit" name="regVare" value="Registrer ny vare"/> <% ArrayList<Vare> varene = new ArrayList<Vare>(); String navn = request.getParameter("navn"); double pris = Double.parseDouble(request.getParameter("pris")); Vare v = new Vare(navn, pris); varene.add(v); %> </body></html> </pre>
--	---

Oppgave a)

Slik koden er laget nå, vil det opprettes et tomt vareregister hver gang nettsiden lastes. Hvilke(n) teknikken vil du bruke for at vareregisteret ikke skal nullstilles for hver gang, og slik at alle kunder som kommer på nettsiden får tilgang til det samme registeret? Se bort fra datafiler og databaser i denne oppgaven.

Merk at vareregisteret ikke må forveksles med en personlig handlekurv.

For at alle brukere skal få tilgang til samme ArrayList kan vi lagre vareregistret i Application-objektet:

Hente: `ArrayList<Vare> varene = (ArrayList<Vare>)application.getAttribute("varene");`
 Opprette nytt dersom ikke finnes: `if (varene == null) varene = new ArrayList<Vare>();`
 Skrive endringer tilbake: `application.setAttribute("varene", varene);`

Dersom nettstedet består av bare én side, kan en bruke en objektvariabel til å lagre data felles for alle besøkende:

```
<%! ArrayList<Vare> varene = new..
```

Oppgave b)

Dessverre så viser det seg at siden ikke virker som den skal – det kastes to unntak når koden kjøres:

"ArrayList cannot be resolved to a type"

"Vare cannot be resolved to a type"

Hva kan være galt, og hvordan kan du løse dette problemet?

Må importere nødvendige pakker:

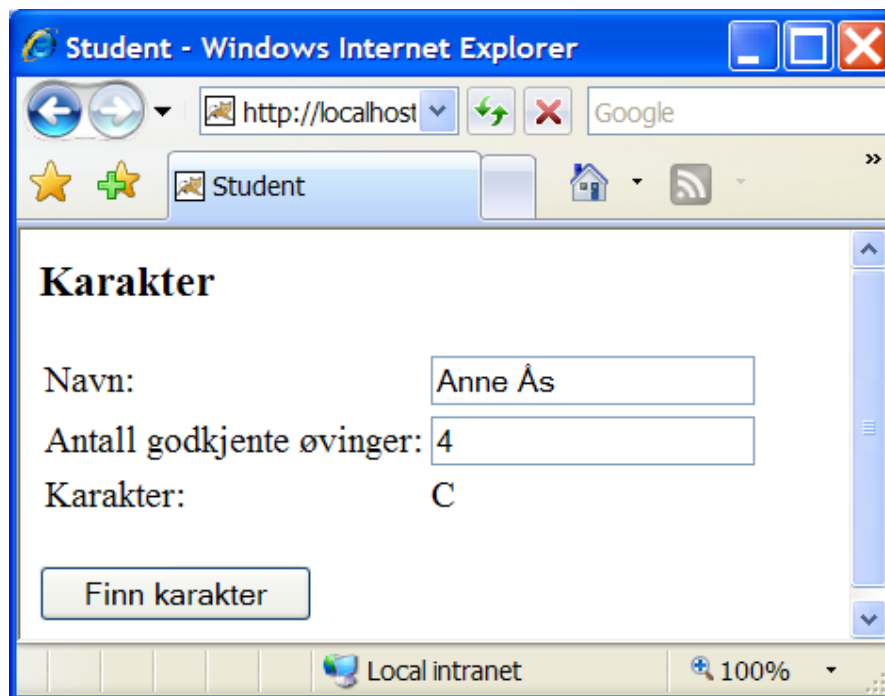
```
<% @ page import="java.util.ArrayList,EksDes2009.Vare" %>
```

Oppgave c)

Du har løst problemet i oppgave b) og får fram skjemaet. Du fyller inn data om ny vare og trykker på knappen "Registrer ny vare". Knappen fungerer ikke. Hvorfor?

Manglende form-tag: `<form action="#" method="post">`

Oppgave 3 (JSF) – vekt 35%



Dette er en meget primitiv applikasjon for beregning av karakter på det øvingsbaserte prosjektet.

Alle deloppgavene forholder seg til denne applikasjonen. Kildeteksten er gitt i vedlegg 1.

Oppgave a)

Forsøk på å kjøre applikasjonen gir følgende feilmelding:

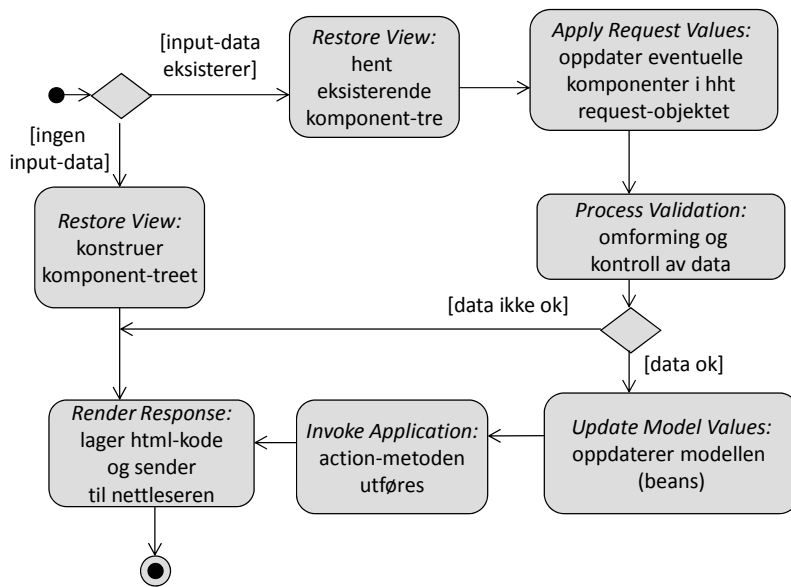
```
javax.servlet.ServletException: /index.jsp(17,41) '#{student.karakter}'  
Property 'karakter' not found on type student.StudentBean  
javax.faces.webapp.FacesServlet.service(FacesServlet.java:277)
```

Forklar hvorfor denne feilmeldingen kommer. Hva må gjøres for at applikasjonen skal fungere?

Må legge inn denne metoden i klassen StudentBean:

```
public char getKarakter() {  
    return karakter;  
}
```

I resten av denne oppgaven skal du, i tillegg til applikasjonen nevnt foran, forholde deg til ulike faser i livssyklusen til en JSF-applikasjon:



Oppgave b)

Fasen "Process Validation" består av to delprosesser: Omforming (converting) og kontroll av data (validation). Forklar helt konkret hva som skjer i de to delprosessene når applikasjonen foran kjører. Relater svaret til jsf-komponentene i index-filen i vedlegg 1.

Alle data i request-objektet er strenger. Delprosessen "omforming" betyr omforming fra strenger til riktig datatype. Datatypen er gitt av set-metoden i den tilhørende bean. I dette eksemplet vil attributtet *antGodkjent* måtte omformes til *int*. Det andre input-attributtet er *navn* med datatype *String* – ingen omforming nødvendig.

Komponentreet bør nevnes, se side 30-33 i boka. De omformede verdiene lagres i såkalte "lokale verdier". Kan hentes ut via metoden `getLocalValue()` i `UIInput`-klassen.

Delprosessen "validering" utføres etter omforming og betyr i dette tilfellet å ta hensyn til:

```
<f:validateLongRange minimum="0" maximum="5"/>
```

Det vil si å kontrollere at *antGodkjent* er i intervallet [0, 5].

Hvis omformingen til *int* ikke går bra, eller hvis det går bra, at valideringen (tallet ligger utenfor området [0, 5]) feiler, følger vi pilen "data ikke ok" på figuren. Siden vises på nytt i nettleseren.

Det er knyttet feilmelding til *antGodkjent*:

```
<h:message for="antall"/>
```

Denne medfører utskrift av en standard feilmelding. Uten `<h:message for="antall"/>` - ingen melding, kun framvisning av siden på nytt.

Oppgave c)

Hva skjer i fasen "Update Model Values"? Relater svaret til koden i vedlegg 1.

Her blir metodene `setNavn()` og `setAntGodkj()` kalt, slik at "the managed bean" blir oppdatert.

Oppgave d)

Både på figuren over og i index-filen i vedlegg 1 refereres det til *action*-attributtet. Det er også et attributt som heter *actionListener*. Hva skiller disse attributtene fra hverandre? Når bør man bruke det ene, og når bør man bruke det andre? Hva er sammenhengen mellom dem?

For det første, her er det ikke snakk om *actionListener* i Swing!

Mesteparten av side 275 i boka må med i svaret her.

Oppgave 4 (databaser og web-applikasjoner) – vekt 20%

I Java-koden kan vi gi tilgang til en databaseforbindelse enten ved å skrive

```
Connection forbindelse = DriverManager.getConnection(databasenavn);
```

eller

```
Connection forbindelse = ds.getConnection(); // ds er et object av klassen DataSource
```

a) Hva skiller disse måtene å hente en databaseforbindelse på?

Her er det aller viktigst å få med forskjellen på å sette opp en forbindelse fra bunnen av (`DriverManager`) og det å låne en forbindelse fra en databasepool. Bra om en husker å fortelle at forbindelsen må lukkes, eventuelt leveres tilbake til poolen, etter bruk.

Det er også fint om kandidaten har med forskjellen mellom oppsettene. Når en bruker `DriverManager` må databasenavn, brukernavn og passord ligge i applikasjonen. Med `DataSource` er ansvaret for dette flyttet til et tjenerprogram, her er det web-tjeneren. Og det er mulig å skifte database uten å endre noe i applikasjonene – så lenge Resource-navnet er det samme.

Noen misforståelser ble oppdaget under rettingen: Noen tror at bruk av `DataSource` krever at databasen kjører på samme maskin som web-tjeneren (eller "lokalt" som de sier, hva de nå

mener med det) – det er ikke nødvendig. Andre blander inn transaksjonshåndtering og flerbrukerproblematikk her. De sier at en kun kan ha én bruker hvis en benytter DriverManager. Til dette er å si at transaksjonshåndtering og flerbrukerproblematikk er *uavhengig* av hvordan databaseforbindelsen er opprettet.

- b) Hvorfor bør vi bruke PreparedStatement-objekter i stedet for Statement-objekter når vi jobber med databaser i web-applikasjoner?

Her er skadelig kode i form av SQL-injeksjon stikkordet. Oppgaven spør ikke etter eksempler, det er derfor ikke nødvendig å komme med det her – selv om det ikke skader. Mange bruker "drop table" som eksempel, men det er neppe mulig å få til i JDBC ettersom det ikke er mulig å sende inn flere sql-setninger av gangen. ... Les for øvrig side 3-6 i leksjon 10.

Vedlegg 1 – kildekode til applikasjon i oppgave 3

index.jsp

```
<!-- Oppgave, eksamen 2009, filnavn: index.jsp -->
<html>
  <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
  <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
  <f:view>
    <head>
      <title>Student</title>
    </head>
    <body>
      <h3>Karakter</h3>
      <p></p>
      <h:form id="eksamen">
        <h:panelGrid columns="2">
          Navn: <h:inputText value="#{student.navn}"/>
          Antall godkjente øvinger:<h:inputText id="antall" value="#{student.antGodkjent}">
            <f:validateLongRange minimum="0" maximum="5"/>
          </h:inputText>
          Karakter:<h:outputText value="#{student.karakter}"/>
        </h:panelGrid>
      <p></p>
      <h:message for="antall"/>
      <p></p>
      <h:commandButton value="Finn karakter" action="#{student.oppdaterKarakter}"/>
    </h:form>
  </body>
</f:view>
</html>
```

faces-config.xml

```
<?xml version="1.0"?>
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd"
  version="1.2">
  <managed-bean>
    <managed-bean-name>student</managed-bean-name>
    <managed-bean-class>student.StudentBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
</faces-config>
```

StudentBean.java

```
package student;

public class StudentBean {
    private String navn = "";
    private int antGodkjent = 0;
    private char karakter = 'F';

    public StudentBean() {
    }

    public String getNavn() {
        return navn;
    }

    public void setNavn(String navn) {
        this.navn = navn;
    }

    public int getAntGodkjent() {
        return antGodkjent;
    }

    public void setAntGodkjent(int antGodkjent) {
        this.antGodkjent = antGodkjent;
    }

    public void oppdaterKarakter() {
        System.out.println("oppdaterKarakter");
        if (antGodkjent >= 4) {
            karakter = 'C';
        } else if (antGodkjent == 3) {
            karakter = 'D';
        } else if (antGodkjent == 2) {
            karakter = 'E';
        }
    }
}
```