



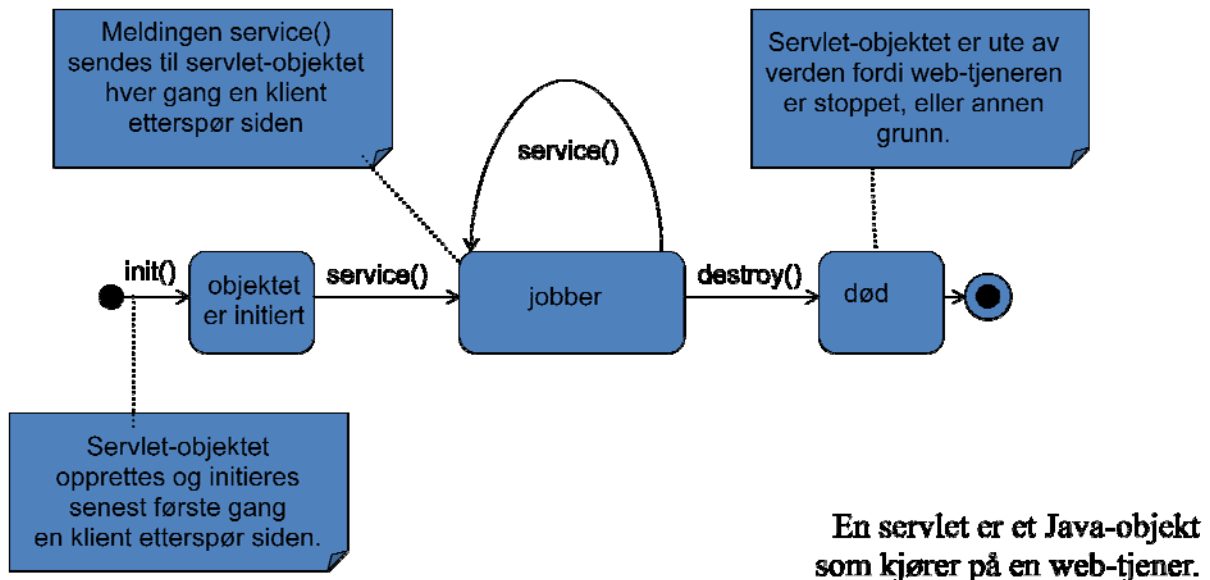
**HØGSKOLEN I SØR-TRØNDELAG**  
Avdeling for informatikk og e-læring - AITeL

**Løsningsforslag**

|   |  |
|---|--|
| <b>Kandidatnr:</b>  |  |
| <b>Eksamensdato:</b>  | 2.desember 2010                                      |
| <b>Varighet:</b>  | 0900-1100  |
| <b>Emnekode:</b>  | LO348D<br>LN349D                                     |
| <b>Emnenavn:</b>  | Web-applikasjoner med JSP og JSF                     |
| <b>Klasse(r):</b>   | 2 ING<br>Nettstudenter                               |
| <b>Studiepoeng:</b>   |  |
| <b>Faglærer(e):</b>   | Else Lervik, Grethe Sandstrak, Mildrid Ljosland      |
| <b>Kontaktperson (adm.)</b>   |  |
| <b>Hjelpemidler:</b>  | Ingen  |
| <b>Oppgavesettet består av:</b>   | 5 oppgaver og 3 sider (inkludert forside og vedlegg) |
| <b>Vedlegg består av:</b>   | 0 sider  |
| <b>Merknad: Oppgaveteksten kan beholdes av studenter som sitter eksamenstiden ut.</b> |  |
| <b>Lykke til!</b>   |  |

## Oppgave 1 (JSP) – vekt 15%

En JSP blir kompilert om til en servlet, som er et Java-objekt som kjører på en web-tjener. Beskriv livsløpet til en servlet. Ta gjerne utgangspunkt i figuren under.



Når en klient etterspør en jsp-nettside vil det sjekkes om dette er første gang nettsiden kalles eller om servlet er eldre enn jsp-siden. Dersom om en av disse testene slår til (første gang siden kalles / siden er endret), vil jsp-siden bli oversatt til en servlet vha JavaServer pages kompilator og deretter kjøres initierings-metoden. Denne kjøres kun en gang.

Etter kompilering og initiering er serlveten klar til å behandle forespørsler (requests). Det opprettes en ny tråd for hver forespørsel. Service-metoden blir kalt og denne behandler forespørselen videre og gir respons til klienten. Service-metoden kjøres så hver gang nettsiden etterspørres (kompilering og initiering av servleten blir da ikke utført)

Til sist i livsløpet til en servlet kalles metoden destroy – har kan man frigjøre evt. ressurser ol. Lik init-metoden kalles også denne metoden kun én gang. Det kan skje ved at web-tjener blir stoppet, eller applikasjonen blir fjernet. Merk: En kan ikke stole 100% på at denne metoden blir kjørt – ved for eksempel server-krasj..

## Oppgave 2 (JSP) – vekt 10%

Inkludering av filer kan være nyttig når vi for eksempel har flere nettsider som skal ha samme logo. Beskriv hvordan vi kan inkludere filer i en JSP på to ulike måter. Få fram ulikhetene mellom de to metodene og hva som kan avgjøre hvilken metode en må bruke.

[Direktivet include/ statisk inkludering.](#)

`<%@ include file="logo.html" %>` - jsp direktiv

Filen som inkluderes blir satt inn i JSP-koden, eksakt der direktivet står i koden FØR JSP'en kompileres til en servlet => Servlet'en inneholder dermed begge filene.

### Dynamisk inkludering

`<jsp:include page="logo.html" flush="true" %>` - jsp-aksjon

Filen som inkluderes blir først inkludert på det tidspunktet forespørselen blir gjort. Servlet'en henter inn den aktuelle fila først når respons på en forespørsel gis.

Brukes når:

- vi ikke har to (eller flere) JSP'er som bruker kode fra hverandre.
- Ikke er veldig opptatt av ytelse. Statisk inkludering vil være raskere da all kode er kompilert inn i samme servlet

## Oppgave 3 (JSP/JSF) – vekt 15%

Forklar kort hva SQL-injeksjon er, og hvordan vi kan beskytte oss mot det?

SQL-injeksjon er injisering av fremmed kode i SQL-setninger. For eksempel kan en logge seg på et nettsted ved å legge inn sql-kode i brukernavn-feltet: `admin' or '1'='1'` og la passordfeltet være blank. Da vil sql-setning se ut feks slik: "

```
select * from bruker where brukernavn ='admin' OR '1'='1' and passord = ' '
```

På denne måten får man en sql-setning som alltid vil være sann og dermed tilgang til nettsider som skulle vært lukket for ikke autoriserte personer.

Hindre SQL-injeksjon

Tålmodige hackere kan være i stand til å avsløre tabellstrukturer og slette data. Detaljerte feilmeldinger til klienten ved syntaksfeil (SQLException) som for eksempel forteller at kolonne X ikke finnes i tabell Y, kan være til stor hjelp for disse menneskene. Det er derfor ingen god idé å sende SQLExceptions til klienten, de hører hjemme på tjenersiden – det er dessuten kun der de kan gis en fornuftig behandling.

Filtrere input, slik at for eksempel apostrofer og andre tegn på "svartelista" stoppes. Et annet alternativ er å bruke PreparedStatement. Denne fungerer slik at SQL-setningen kompileres på forhånd. Parametrene sendes inn til ferdig compilert kode. Dermed unngår vi at databasetjeneren mottar en tekststreng med en fullstendig SQL-setning (inkludert evt. injisert kode). Dersom samme sql skal utføres mange ganger i en applikasjon så får man også en ytelsesgevinst ved bruk av `prep.stmt`.

Eks `prep.stat`: `String sql = "UPDATE Student SET navn = ? WHERE studid= ?"` Spørsmålstegnene representerer plassholdere. Ved kjøring av sql-setningen vil kun verdiene oppdateres. Merk: Verdiene blir validert.

## Oppgave 4 (JSF) – vekt 30%

Vi har en bean-klasse som vi har gitt navnet Eksamen2010 med metoden `public ArrayList<Person> getPersonene()`. Klassen Person består av metodene `getNavn()`, `getAdresse()` og `getAlder()`.

Vi ønsker å få vist fram persondataene i en tabell.

- a) Forklar hva som skjer hvis vi skriver

```
<h:dataTable value="#{Eksamen2010.personene}" var="person" >
  <h:column>
    #{person.navn}
  </h:column>
  <h:column>
    #{person.adresse}
  </h:column>
  <h:column >
    #{person.alder} år"
  </h:column>
</h:dataTable>
```

Det lages en tabell med tre kolonner. Data til tabellen hentes fra beanen Eksamen2010, der egenskapen personene (metoden `getPersonene`) blir brukt. For hvert objekt i tabellisten, lages en linje i tabellen. Vi henviser til et slikt objekt med navnet person. I første kolonne skrives peronens avn ut, i andre kolonne adressen og i tredje kolonne alderen. PS trykkfeil: Skal ikke stå " etter år

- b) Nederst i tabellen vil vi ha utskrevet

Gjennomsnittsalder: xxx,x år

(Gjennomsnittsalderen regnes ut ved å summere alle aldrene, og deretter dele på antallet.)

Skisser,gjerne med kode, de endringene som er nødvendig i koden fra a. Må noe endres/legges til i java-klassene, i såfall hva og hvor?

I første eller andre `<h:column>`:

```
<f:facet name="footer">
```

Gjennomsnittsalder:

```
</f:facet>
```

I tredje `<h:column>`:

```
<f:facet name="footer">
```

```
<h:outputText value="#{Eksamen2010.gjennomsnitt}">
```

```
<f:convertNumber pattern="0.0" />
```

```
</h:outputText>
```

år

</f:facet>

I Eksamen2010: Må lage en metode getGjennomsnitt() som finner gjennomsnittet av alderen på alle personene.

- c) Forklar hva du vil gjøre for å få kolonnen med alder til å vises med rød skrift. Angi helst flere ulike måter det kan gjøres på.

Må definere stil for kolonne-teksten. Dette kan gjøres på diverse måter:

Alt 1: <div style="color:red"> #{person.alder} år </div>

Alt 2: <h:outputText value=" #{person.alder} år" style="color:red" />

Alt 3: <h:outputText value=" #{person.alder} år" styleClass="red" /> pluss stildefinisjon

```
.red {
```

```
  color: red;
```

```
}
```

Alt 4: <h dataTable columnClasses="empty, empty, red" > pluss stildefinisjonene

```
.empty {
```

```
}
```

```
.red {
```

```
  color: red;
```

```
}
```

PS: <column> har ikke noen attributt som kan brukes

- d) Vi ønsker at bare de personene som er myndige (18 år eller eldre) skal vises fram. Hvordan vil du løse dette?(Du kan se bort fra problemet med at da bør gjennomsnittsalderen endres).

Alternativ 1:

Kan lage en egenskap isMyndig() i Person-klassen. Alle kolonnetekstene legges inn i en h:outputText, og attributtet rendered legges til.

For navnet blir det slik:

```
<h:outputText value="#{person.navn}" rendered="#{person.myndig}"/>
```

Tilsvarende for adresse og alder.

Eventuelt kan egenskapen `isMyndig()` sløyfes, isteden skriver vi `rendered="#person.alder>=18"`

Alternativ 2:

Lage en ny metode `finnAlleOver18` eventuelt `finnAlleOver(int alder)` i `Eksamen2010` som returnerer en `ArrayList` med de ønskede personene, og bruke den som value i `dataTable`.

## Oppgave 5 (JSF) – vekt 30%

Et web-sted består av en rekke sider. Forklar hvordan du vil løse det følgende (skriv noen få linjer om hvert punkt):

- a) Alle sidene skal ha et enhetlig preg.

Bruker Facelets. Det innebærer at vi lager en template som definerer grunnstrukturen på sidene. Her setter vi blant annet opp hvilke stilark vi skal bruke, og vi definerer "plassholdere" med `ui:insert`. I enkeltsidene bruker vi `ui:composition` til å henvise til templatene, og fyller inn "plassholderne" ved hjelp av `ui:define`. Det som vises fram, er templatene med de fylte plassholderne.

- b) Noen av sidene skal passordbeskyttes.

Vi bruker web-tjeneren til å foreta autentisering. Det innebærer at vi må definere en database med tabeller over roller og brukere og en autentiseringsrealm. De sidene som skal passordbeskyttes, legges i en egen katalog. I `web.xml` setter vi opp hva den beskyttede katalogen heter, hvilke roller som er definert samt hvilken type autentisering vi ønsker.

- c) Sidene skal ha språkstøtte for norsk og engelsk

Alle tekster må legges i en property-fil. Denne filen må finnes i to utgaver, en som slutter på `_no` (norske tekst), og en som slutter på `_en` (engelske tekst). I `faces-config.sys` må vi sette opp en resource-bundle som forteller hvor disse property-filene finnes, og hva vi ønsker å kalle dem i `jsf`-filene. Dessuten må vi fortelle hvilke språk som er støttet, det gjøres også i `faces-config.xml`. I `jsf`-filene henviser vi til disse filene ved å skrive for eksempel `#{tekst.LoggInnVerdi}` der `tekst` er navnet vi har definert på property-filene mens `LoggInnVerdi` er navnet på en property. Når `jsf`-filen vises fram, hentes verdien på denne propertyen fra riktig språkversjon.

Vi kan la brukeren velge språk ved å ha en knapp eller lignende som fører til at den riktige Locale-verdien settes i Context-objektet.

- d) På en av sidene skal brukeren oppgi et bankkontonummer. Dette *må* oppgis, og det må være nøyaktig 11 siffer. Bruker skal få beskjed hvis dette ikke er oppfylt.

Må legge inn validering på dette feltet i `jsf`-filen. For å sjekke at noe er skrevet, brukes `required="true"`. For å sjekke at det er nøyaktig 11 siffer, kan man bruke `validateRegex pattern="\d{11}"` eller man kan bruke `validateLongRange minimum="10000000000" maximum="99999999999"`. I siste tilfelle må variabelen være av typen `long`. Og så må man legge inn enten `<h:message for="kontonummer" />` eller `<h:messages />` for å få feilmeldingen ut til bruker .