

# **EKSAMENSOPPGAVE**

## **Løsningsforslag**

**Emnekode: DAT108**

**Emnenavn: Programmering og webapplikasjoner**

**Utdanning/kull/klasse: 2. klasse data/inf**

**Dato: 15. juni 2021**

---

Eksamensform: Skriftlig hjemmeksamen (Wiseflow | FLOWassign)

Eksamenstid: 4 timer (0900-1300) + 0,5 timer ekstra for innlevering

Antall eksamensoppgaver: 5

Antall sider (medregnet denne): 14

Antall vedlegg: Ingen

Tillatte hjelpemiddel: Alle.

Lærere:      Lars-Petter Helland (928 28 046) lph@hvl.no  
                 Bjarte Kileng (909 97 348) bki@hvl.no

## Oppgave 1 (16% ~ 38 minutter) – Lambda-uttrykk og strømmer

For å få full score må løsningene ikke være unødige kompliserte.

Vi har definert et interface med en metode som tar inn en streng og returnerer en annen streng, f.eks. som returnerer en endret utgave av den som kom inn:

```
@FunctionalInterface
interface Stringfunksjon {
    String anvend(String inn);
}
```

Vi ønsker å lage en metode for å skrive ut tekst formatert, slik:

```
/**
 * Metoden endrer på strengen som kommer inn v.hj.a. Stringfunksjon format,
 * og skriver deretter resultatet ut på skjermen (med System.out.print).
 */
private static void skrivUtFormatert(String inn, Stringfunksjon format) {
    //Her mangler litt kode - Oppgave 1a)
}
```

a) Skriv koden som mangler i kodelistingen over.

Forslag til løsning m/kommentarer:

```
public static void skrivUtFormatert(String inn, Stringfunksjon format) {
    String ut = format.anvend(inn);
    System.out.print(ut);
}
```

evt.

```
public static void skrivUtFormatert(String inn, Stringfunksjon format) {
    System.out.print(format.anvend(inn));
}
```

Ikke så mye mer å si om denne.

b) Skriv en **main**-metode som bruker `skrivUtFormatert()` til å skrive ut en tekst (som du velger selv) med

- i. Kun store bokstaver, eks. (Java) -> **J A V A**
- ii. Innrammet, eks. (Java) -> **[Java]**
- iii. Med blank mellom hvert tegn -> **J a v a**

Bruk lambda-uttrykk til å representere Stringfunksjon-objektene, og tilordne dem helst til variabler med gode navn før de brukes.

Forslag til løsning m/kommentarer:

```
public static void main(String[] args) {  
  
    String tekst = "Java";  
  
    //b.i  
    Stringfunksjon storeBokstaver = x -> x.toUpperCase(new Locale("no"));  
    skrivUtFormatert(tekst, storeBokstaver);  
  
    //b.ii  
    Stringfunksjon innrammet = x -> "[" + x + "]";  
    skrivUtFormatert(tekst, innrammet);  
  
    //b.iii  
    Stringfunksjon blankeMellom = x -> x.replaceAll("", " ");  
    skrivUtFormatert(tekst, blankeMellom);  
  
    //b.iii - alternativ  
    Stringfunksjon blankeMellom = x -> {  
        String ny = "";  
        for (char tegn : x.toCharArray()) {  
            ny += tegn + " ";  
        }  
        return ny;  
    };  
    skrivUtFormatert(tekst, blankeMellom);  
}
```

Kommentarer:

- At jeg bruker `toUpperCase` med norsk `Locale` gjør at det også virker for æøå.

- c) Skriv en metode **kombiner(...)** som tar inn to Stringfunksjon-er og returnerer en Stringfunksjon som gir en kombinasjon av de som puttes inn. F.eks. kan dette brukes til å gi en utskrift som er en kombinasjon av store bokstaver og blanke mellom hvert tegn (J A V A).

Forslag til løsning m/kommentarer:

```
public static Stringfunksjon kombiner(Stringfunksjon f1, Stringfunksjon f2) {  
    return x -> f2.anvend(f1.anvend(x));  
}
```

Tanken her er at f1 og f2 anvendes etter tur på x.

Man kan jo selvfølgelig argumentere for at oppgaven gir mulighet for tolkning, f.eks. om f1 skal anvendes før f2 eller omvendt. En del f1-er og f2-er vil jo også kunne motvirke hverandre, f.eks. til store bokstaver og til små bokstaver. ...

Eksempel på anvendelse av løsningen:

```
Stringfunksjon storeOgBlanke = kombiner(storeBokstaver, blankeMellom);  
skrivUtFormatert(tekst, storeOgBlanke);
```

- d) Anta nå at du har en liste **spraakliste** som inneholder navn på en del programmeringsspråk. Skriv kode som gir en utskrift av disse språkene, bortover på én linje, innrammet (ref. b.ii). Eks: **[Java][C#][Haskell][Rust]**. Dette skal gjøres på to ulike måter:

- i. Ved å bruke streams-APIet, men uten å bruke skrivUtFormatert()
- ii. Ved å bruke skrivUtFormatert() (så enkelt som du klarer)

Forslag til løsning m/kommentarer:

```
List<String> spraakliste = Arrays.asList("Java", "C#", "Haskell", "Rust");  
  
//d.i  
spraakliste.stream().map(x -> "[" + x + "]").forEach(System.out::print);  
//evt.  
spraakliste.stream().forEach(x -> System.out.print "[" + x + "]" ));  
  
//d.ii - Bruker Stringfunksjon-variabelen innrammet fra b.ii  
spraakliste.forEach(s -> skrivUtFormatert(s, innrammet));  
//evt.  
spraakliste.forEach(s -> skrivUtFormatert(s, x -> "[" + x + "]" ));
```

Ikke så mye mer å si om denne.

## Oppgave 2 (8% ~ 20 minutter) – Passordsikkerhet

Når vi lagrer passord **salter**, **hasher** og **itererer** vi for å generere det som skal lagres.

- a) Hva er hashing? Hvorfor hasher vi i stedet for å bruke vanlig kryptering?
- b) Hva er salting? Hvorfor salter vi?
- c) Hva er iterering (key stretching)? Hvorfor itererer vi?

Forslag til løsning m/kommentarer:

### a) Hashing

Hashing (kryptografisk hashing) er en enveis avbildning av en verdi til en annen verdi. Det er ikke mulig å avlede den opprinnelige verdien ut fra den avledete. Noen egenskaper:

- Det er umulig å gjette seg til en hash. Nesten lik input gir helt ulik output.
- Hashverdier har samme størrelse (antall bits) uavhengig av størrelse på input.
- Ulike inputs gir ulike outputs. Det finnes «få» kollisjoner.

Vi foretrekker hashing over kryptering til passord i hovedsak av to grunner:

- Passord er sensitive data, og man ønsker ikke at NOEN skal kunne se disse.
- Kryptering er mer sårbart for hacking siden man kun trenger å finne nøkkel for å dekryptere. Hashing har ingen nøkkel, og man MÅ derfor prøve seg frem (tar tid).

### b) Salting

Salting er det å legge til noe data i tillegg til det som skal hashes før man hasher. Et salt bør ha en viss størrelse, være tilfeldig generert, og være unikt per bruker/passord.

Vi bruker salting til passord siden vi ønsker å unngå at samme passord gir samme hash. Uten salting kan hash til milliarder vanlige passord beregnes og lagres en gang for alle slik at det er lett å finne passord som hører til en gitt hash. Eller hvis du kjenner et passord, så vil andre med samme hash ha samme passord. Med salting vil alle passord ha ulik hash, basert på passord+salt, slik at man ikke kan utlede passord fra gitt hash.

### c) Iterering

Angrepsteknikker for å finne passord innebærer å prøve seg frem (enten brute-force eller via ordliste). For å gi økt beskyttelse ønsker vi å gjøre hashingen treig. Formålet med iterering, dvs. å hashe mange ganger i stedet for én, er å gjøre hashingen treig, f.eks. 1000 ganger treigere enn om man kun trengte å beregne én hash per forsøk. Key stretching er en samlebetegnelse på teknikker som har det nevnte som formål.

### Oppgave 3 (20% ~ 48 minutter) – JavaScript

a) Teksten under blir ofte brukt i JavaScript-kode:

```
"use strict";
```

i. Hva er konsekvensene av å legge denne teksten i JavaScript-koden?

Løsning:

- Skrivefeil i variabelnavn vil gi JavaScript feil.
  - Uten strict mode blir istedet en ny variabel opprettet.
- Uten strict mode blir mange feil ignorert uten noen melding, f.eks.:
  - Forsøk på tilordning til ikke skrivbar egenskap.
  - Forsøk på å slette en ikke slettbar egenskap.
- Reserverte ord som *static*, *eval* og *arguments* kan ikke brukes som variabler.
- Endel ord som kan bli brukt i fremtidige JavaScript-versjoner blir ulovlig som variabelnavn, f.eks. *public*, *private*, *protected*, *interface*.
- Bruk av *with* konstruksjonen er ulovlig.
- Hvis metode kjøres utenfor kontekst av et objekt er *this* udefinert.

ii. Hvor i JavaScript-koden skriver vi denne teksten?

Løsning:

I starten av JavaScript-koden for å gjelde hele koden, eller i starten av en funksjon for å gjelde for koden inne i funksjonen.

b) Nøkkelordet *this* og mistet kontekst.

i. I noen situasjoner vil *this* miste konteksten av objektet som eier metoden som bruker *this*. Gi eksempler på situasjoner når dette skjer, og forklar årsaken.

Løsning:

Størrelsen *this* mister sin kontekst hvis en metode kjøres via en referanse. I DAT108 har vi kun sett på dette i forbindelse med hendelseshåndterere.

```
elmRef.addEventListener(hendelse,objekt.metode);
```

Her er *objekt.metode* en referanse til metoden. Når hendelsen skjer, kjøres metoden via referansen, og da er konteksten av *objekt* borte. Istedet vil *this* være *elmRef*.

- ii. Gi eksempler på løsninger for å bevare konteksten ved bruk av *this*.

Løsning:

I DAT108 brukte vi *bind* for å binde konteksten:

```
elmRef.addEventListener("click",objekt.metode.bind(objekt));
```

En annen vanlig løsning er å bruke en omsluttende funksjon:

```
elmRef.addEventListener("click",e => {objekt.metode(e)});
```

- iii. I noen situasjoner kan bruk av pil-syntaks for funksjoner løse problemer knyttet til bruk av *this*. Vis med eksempel hva som er problemet og forklar hvordan pil-syntaks kan løse problemet.

Løsning:

Funksjoner lager sin egen binding av *this*, normalt til objektet som har funksjonen som metode.

Funksjon laget med pilsyntaks lager ingen egen binding av *this*, men bruker det omsluttende miljøet sin *this*.

Anta at koden under finnes i en metode til et objekt, og at *m* er en annen metode i objektet:

```
elmRef.addEventListener("click",function(e) {this.m(e)});
```

Funksjonen over lager sin egen binding av *this*, til *elmRef* som trigget hendelsen. Koden feiler da *this.m* forsøker å kjøre *elmRef.m*.

I koden under lages ingen egen binding av *this*. Verdien til *this* tas fra det omsluttende miljøet, og vil da være objektet som eier metoden med koden.

```
elmRef.addEventListener("click",(e) => {this.m(e)});
```

- c) Opprett en JavaScript klasse **TekstAnalyse**. Klassen har følgende metoder som kan brukes utenfor klassen (brukes **public**):

- Setter *tekst*. Du kan i stedet bruke en metode *setTekst* hvis du ikke vet hvordan du lager en setter.
- Getter *liste*. Du kan i stedet bruke en metode *getListe* hvis du ikke vet hvordan du lager en getter.
- Getter *antallord*. Du kan i stedet bruke en metode *getAntallord* hvis du ikke vet hvordan du lager en getter.
- Metode *finnhypigsteord*.

Nedenfor omtales *getterne*, *setterne* og metodene, med tips om hvordan oppgaven kan løses.

#### Setter *tekst* (evt. metode *setTekst*)

Denne registrerer en tekst av ord som det skal arbeides med videre i klassen.

#### Getter *liste* (evt. metode *getListe*)

Denne returnerer en **Array** forekomst av alle ord i teksten. Samme ord kan forekomme flere ganger hvis det opptrer flere ganger i teksten.

Metoden *match* på forekomster av **String** lar oss enkelt fylle en **Array** med ord fra en tekst. Metoden tar et regulært uttrykk som argument og returnerer en **Array** av de delene av teksten som passer med mønsteret. Et uttrykk som kan brukes for denne oppgaven er:

- `/\p{Letter}+/ug`

#### Getter *antallord* (evt. metode *getAntallord*)

Denne returnerer antall ord i teksten. Samme ord telles flere ganger hvis det opptrer flere ganger i teksten.

#### Metode *finnhyppigsteord*

Denne metoden skal returnere de ordene som forekommer fleste antall ganger i teksten, og antall ganger disse ordene forekommer.

Observer at forskjellige ord kan forekomme like mange ganger, så derfor må metoden returnere en **Array** forekomst av ord. Returverdien skal være følgende objekt:

```
{"antall": maksantall, "ord":ordliste}
```

Her er *maksantall* det største antall ganger et ord forekommer i teksten, og *ordliste* er en **Array** av ord.

Metoden skal ikke skille på store og små bokstaver når ord telles.

For å finne antall forekomster av hvert enkelt ord kan du bruke en **Map**, der nøkkel er ordet, og verdi er antall forekomster av ordet. En **Map** sin metode *has* kan sjekke om en nøkkel finnes, og metodene *get* og *set* brukes for å hente ut og legge til et element til en forekomst av **Map**.

For å unngå å skille på store og små bokstaver må ordene konverteres når de legges inn i en **Map**, f.eks. ved å bruke metoden *toLocaleUpperCase* som kan brukes på forekomster av **String**.

#### Kodeeksempler

Eksempelet under oppretter en **TekstAnalyse** og legger inn en tekst:

```
const analyse = new TekstAnalyse();
analyse.tekst = "Velkommen, ja velkommen til eksamen.";
```

Nedenfor vises bruk av getteren *antallord*:



```
console.log(`Antall ord i teksten er ${analyse.antallord}`);
```

Under vises utskriften i nettleserkonsollet fra koden over:

```
Antall ord i teksten er 5
```

Getteren *liste* skal returnere følgende **Array** for den gitte teksten:

```
[ "Velkommen", "ja", "velkommen", "til", "eksamen" ]
```

Metoden *finnhyppestord* skal returnere følgende **Object** for den gitte teksten:

```
{
  "antall": 2,
  "ord": [ "VELKOMMEN" ]
}
```

Med *ordmap* en **Map** av ord kan vi legge til et nytt ord fra variabel *nyttord* med følgende kode:

```
ordmap.set(nyttord, 1);
```

I koden ovenfor er verdien satt til 1 siden dette var et nytt ord, og ordet nå er registrert en gang.

## Oppgave

Skriv JavaScript-koden for **TekstAnalyse** i samsvar med teksten over.

Løsning:

```
class TekstAnalyse {
  _ordliste = null;
  _statistikk = new Map();

  /**
   * @public
   * @param{String} innhold
   */
  set tekst(innhold) {
    /**
     * Kunne opprettet Array og Map i de tilsvarende metodene,
     * men da gjøres det hver gang metodene kjøres.

     * Velger istedet å gjøre det her. Ulempen er at da blir Map
     * og Array opprettet også om de ikke blir brukt.

     * Kunne alternativt sjekket ved bruk om Array og Map
     * finnes, hvis ikke, lag dem.
     */
    this._ordliste = innhold.match(/\p{Letter}+/ug);
    this._statistikk.clear();
    this._ordliste.forEach(
      ord => { this._registrer(ord) }
    );
  }
}
```

```

get liste() {
  return this._ordliste;
}

get antallord() {
  return this._ordliste.length;
}

finnhyppestord() {
  const ord = [];
  const maksantall = Math.max(...this._statistikk.values());
  this._statistikk.forEach(
    (value, key) => {
      if (value === maksantall) {
        ord.push(key);
      }
    }
  );
  return { antall: maksantall, ord: ord };
}

_registrer(ord) {
  ord = ord.toLocaleUpperCase();
  if (this._statistikk.has(ord)) {
    const count = this._statistikk.get(ord);
    this._statistikk.set(ord, count + 1);
  } else {
    this._statistikk.set(ord, 1);
  }
}
}

```

## Oppgave 4 (40% ~ 96 minutter) – Webapplikasjoner, tjenerside

### Innledning

Etter en forelesning lurer ofte foreleser på hvordan det har gått. Synes studentene forelesningen var god? Var det noe som var uklart? Hva kunne vært gjort annerledes? Studenten har kanskje tilsvarende behov: Å gi tilbakemelding, stille spørsmål, si fra om noe som var uklart.

Vi ønsker derfor å lage en web-applikasjon der studenter enkelt kan gi tilbakemelding etter en forelesning (eller en lab-time). Siden for tilbakemelding kan for eksempel se slik ut:



Tilbakemelding på forelesning

localhost:8080/dat108v21exam/tilbakemelding

Søndag 1. november 2020, kl.13:00

Du er registrert som **lph** i klasse **INF\_2019\_HØST**.

Aktuell aktivitet er **DAT108-forelesning kl. 10-12**.

Din tilbakemelding:

☐ ☐ ☐

Skriv din tilbakemelding her

Send

fig. 4.1

**NB! Det skal ikke være innlogging i tradisjonell forstand.**

En (student) bruker blir identifisert ved at en persistent cookie er lagret i nettleseren. Denne blir lagret ved registrering (se neste side), og cookien inneholder brukeren sitt nick.

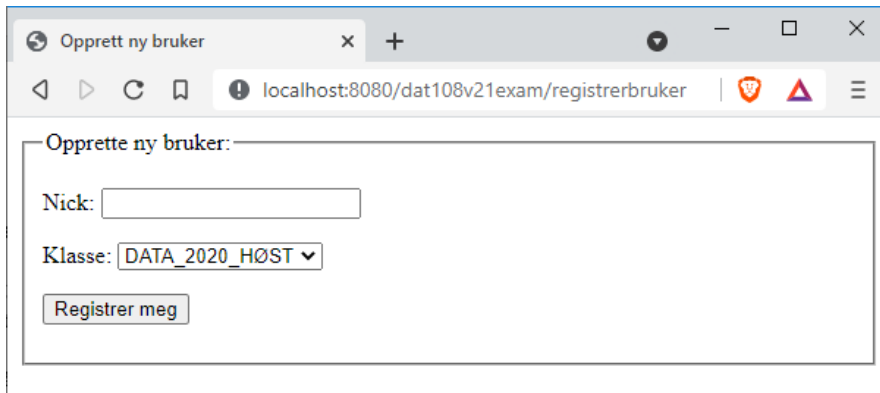
Det er gjort slik for at det skal vere enklere å bruke applikasjonen.

Vi skal ikke se på hele applikasjonen i denne eksamensoppgaven, kun et par utvalgte sider og brukstilfeller.

## Brukstilfelle – Opprette ny bruker (for en student)

Man må registrere seg som bruker for å kunne gi tilbakemelding. Dette blir gjort for å koble student til klasse slik at rett aktivitet (fra timeplanen) kommer frem til rett tid.

Skjema for registrering kan for eksempel se slik ut:



The screenshot shows a web browser window with the title 'Opprett ny bruker'. The address bar shows 'localhost:8080/dat108v21exam/registrerbruker'. The form itself is titled 'Opprette ny bruker:' and contains a 'Nick:' text input field, a 'Klasse:' dropdown menu with 'DATA\_2020\_HØST' selected, and a 'Registrer meg' button.

fig. 4.2

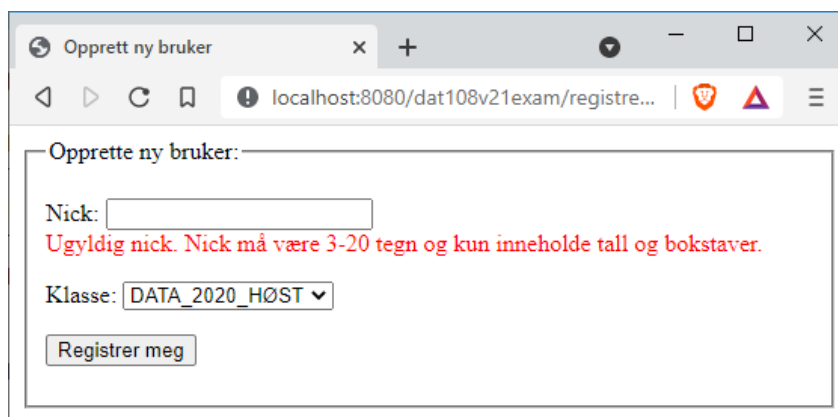
Rullegardin-listen av klassekoder er hentet fra en database (via **StudentFeedbackDAO**).

Når man registrerer seg (og alt går fint), blir student (nick + klassekode) lagret i databasen (via **StudentFeedbackDAO**). Nick-et blir også sendt til nettleser som en persistent cookie (via **CookieUtil**). Ved senere bruk (fra samme nettleser) blir bruker automatisk identifisert via denne cookien.

Hvis registreringen går fint, blir bruker sendt til tilbakemeldingssiden (fig. 4.1).

Feilsituasjoner ved registrering:

- Ved oppretting av bruker skal **nick** valideres. Et nick må være 3-20 tegn langt og kan kun inneholde tall og bokstaver. Ved ugyldig nick skal bruker sendes tilbake til samme side, og en feilmelding skal vises (fig. 4.3).
- Ved oppretting av bruker kan **nick** ikke allerede eksistere i databasen. Ved allerede eksisterende nick skal bruker sendes tilbake til samme side, og en annen feilmelding skal vises.



The screenshot shows the same registration form as in fig. 4.2, but with an error message displayed in red text below the 'Nick:' input field: 'Ugyldig nick. Nick må være 3-20 tegn og kun inneholde tall og bokstaver.' The 'Klasse:' dropdown menu still shows 'DATA\_2020\_HØST' and the 'Registrer meg' button is still present.

fig. 4.3

## Brukstilfelle – Gi tilbakemelding

Fra hovedsiden (fig. 4.1) kan en bruker gi tilbakemelding på en aktivitet for sin klasse.

Tekst i uthevet skrift på siden (fig. 4.1) er dynamiske data.

- Pent formatert dato og klokkeslett kommer fra en **TidOgDatoUtil**.
- Studenten sitt nick ("lph") kommer fra nettleser-cookie, som kan hentes via **CookieUtil**, men må også finnes i databasen, representert ved **StudentFeedbackDAO**.
- Klassekode er koblet til nick, og blir hentet fra databasen via **StudentFeedbackDAO**.
- Aktuell aktivitet (se fig. 4.1) kommer fra **StudentFeedbackDAO**, og er beregnet ut fra gjeldende tidspunkt og timeplan for aktuell klasse.

Feilsituasjoner:

- Hvis man prøver å nå denne siden uten at man har en cookie med et gyldig og registrert nick, skal man omdirigeres til registreringsskjemaet. (Da må man registrere seg på nytt. NB! Man kan ikke registrere seg med samme nick flere ganger / i flere nettlesere.)

Ved trykking på Send-knappen i fig. 4.1:

- **Hva som skjer ved sending av tilbakemelding er ikke en del av denne eksamensoppgaven.**

## Klasser og jsp-er i applikasjonen:

### @Entity

**Student.java** representerer en studentbruker. Har følgende egenskaper:

- @Id String nick;
- String klassekode;

### @Entity

**Aktivitet.java** representerer en aktivitet. Har følgende egenskaper:

- ... (egenskapene er ikke nødvendige for eksamensoppgaven)

og følgende metode:

- String getVisningstekst() // Denne kan brukes i JSP-siden som tekst for gjeldende aktivitet

### @Stateless

#### **StudentFeedbackDAO.java**

StudentFeedbackDAO er en stateless session bean, og skal derfor brukast som en @EJB i servletene. StudentFeedbackDAO bruker JPA til å kommunisere med den underliggende databasen.

- Student hentStudentMedNick(String nick); //null om ingen med dette nick
- Aktivitet hentAktuellAktivitetForStudent(Student student); //null om ingen aktivitet
- List<String> hentListeAvKlassekoder(); //aldri tom
- void registrereStudent(Student student);
- void lagreTilbakemelding(Tilbakemelding tilbakemelding);

#### **TidOgDatoUtil.java**

- String getNaaSomString() // Denne kan brukes i JSP-siden som tekst for dato + tidspunkt

#### **CookieUtil.java**

// Henter cookie fra requesten, dekode og returnerer den som en string

- String getCookieFraRequest(HttpServletRequest request, String cookieNavn)

// Koder cookieverdi, merker for lagring i ett år og legger til responsen

- void leggCookieTilResponse(Response response, String cookieNavn, String cookieVerdi)

#### **NickValidator.java**

// Et nick må være 3-20 tegn langt og kan kun inneholde tall og bokstaver.

- boolean erGyldigNick(String nick); //Sjekker om gyldig nick. Sjekker også != null.

**RegistrerBrukerServlet.java** er knyttet til URL-en **/registrerbruker**.

- **GET** – Ein forespørsel om å få se siden for brukerregistrering, ref. fig. 4.2. Samarbeider med viewet **RegistrerBrukerSkjema.jsp**. Hvis siden skal inneholde en feilmelding er type feilmelding gitt som en parameter **feilkode** til forespørselen: 1=ugyldig nick, 2=nick finst fra før.
- **POST** – Ein forespørsel om å registrere en ny bruker. Hvis alt går bra skal man til slutt omdirigeres til **tilbakemelding**. Hvis ikke alt går bra, skal man omdirigeres til **registrerbruker** igjen.

**GiTilbakemeldingServlet.java** er knyttet til URL-en **/tilbakemelding**.

- **GET** - Ein forespørsel om å få se siden for tilbakemeldingsskjema, ref. fig. 4.1. Samarbeider med viewet **Tilbakemeldingsskjema.jsp**. Alternativt scenario:
  - Hvis man ikke har en cookie med et gyldig nick for ein registrert bruker, skal man omdirigeres til **registrerbruker**.
- **POST** - Ein forespørsel om å registrere en tilbakemelding. **Ikke en del av eksamensoppgaven.**

### **RegistrerBrukerSkjema.jsp**

JSP for å generere sider tilsvarende fig. 4.2 uten, alternativt fig. 4.3 med feilmeldinger.

### **Tilbakemeldingsskjema.jsp**

JSP for å generere sider tilsvarende fig. 4.1. **Ikke en del av eksamensoppgaven.**

## **Krav til løsningen**

For å få full score må du løse oppgaven på best mulig måte i hht. prinsippene i kurset, dvs. **Model-View-Controller, Post-Redirect-Get, EL** og **JSTL** i JSP-ene, **trådsikkerhet, robusthet, ufarliggjøring** av brukerinput, **unntakshåndtering, god bruk av hjelpeklassene, elegant kode**, osv ...

Løsningen trenger ikke å være robust mot «hacking», dvs. utilsiktet bruk.

Løsningen trenger ikke å være robust mot tekniske databasefeil.

## Nå til selve oppgavene

- a) Skriv doGet()-metoden i RegistrerBrukerServlet.java.

Forslag til løsning m/kommentarer og poenggiving (**10p** totalt er maks):

Merknader: Request-parameteren **feilkode** er evt. satt i fbm. redirect til siden ved tidligere feilsituasjon. Antar at **studentFeedbackDAO** er en eksisterende @EJB i servleten.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
```

```
2p*    String feilkode = request.getParameter("feilkode");
```

```
        if (feilkode != null) {
```

```
1p            if (feilkode.trim().equals("1")) {
                request.setAttribute("feilmelding",
                    "Ugyldig nick. Nick må være 3-20 tegn og kun inneholde ...");
```

```
            }
            if (feilkode.trim().equals("2")) {
1p                request.setAttribute("feilmelding",
                    "Ugyldig nick. Nick er allerede i bruk.");
```

```
            }
        }
```

```
2p**    request.setAttribute("klassekoder",
```

```
2p        studentFeedbackDAO.hentListeAvKlassekoder());
```

```
        request.getRequestDispatcher(
```

```
2p        "WEB-INF/RegistrerBrukerSkjema.jsp").forward(request, response);
    }
```

\* Feilhåndtering kan gjøres litt annerledes enn dette uten at det blir feil.

\*\*JSP-en trenger klassekoder til select-boksen. Hvis man ikke henter de her i controlleren, må det argumenteres for og vises hvordan disse hentes et annet sted. Hvis de hentes i JSP-en må jo dette gjøres på riktig måte for å virke (<%@ page import ...%> + <% @EJB ...%> + <%= %> + <c:set.../> ++), og for å få full score.



b) Skriv RegistrerBrukerSkjema.jsp

Forslag til løsning m/kommentarer og poenggiving (10p totalt er maks):

Merknad: Attributtene **feilmelding** og **klassekoder** antas satt i controlleren (Oppgave 4a).

```
<!DOCTYPE html>
<html>
...
<body>
2p    <form method="post">
        <fieldset><legend>Opprette ny bruker:</legend>

1p        <p>Nick: <input type="text" name="nick"><br />
1p        <font color="red">${feilmelding}</font></p>

        <p>Klasse:
1p        <select name="klassekode">
2p + 1p        <c:forEach items="${klassekoder}" var="k">
1p        <option value="${k}">${k}</option>
        </c:forEach>
        </select></p>

1p        <p><input type="submit" value="Registrer meg" /></p>

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

c) Skriv doPost()-metoden i RegistrerBrukerServlet.java.

Forslag til løsning m/kommentarer og poenggiving (10p totalt er maks):

Merknad: Request-parametrene **nick** og **klassekode** er gitt/valgt av bruker, se RegistrerBrukerSkjema.jsp (Oppgave 4b).

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    final int UGYLDIG_NICK = 1;
    final int OPPTATT_NICK = 2;

1p    String nick = request.getParameter("nick");
1p    if (!new NickValidator().erGyldigNick(nick)) {
1p        response.sendRedirect("registrerbruker?feilkode=" + UGYLDIG_NICK);
1p    } else if (studentFeedbackDAO.hentStudentMedNick(nick) != null) {
        response.sendRedirect("registrerbruker?feilkode=" + OPPTATT_NICK);
    } else {
1p        String klassekode = request.getParameter("klassekode");
1p        Student student = new Student(nick, klassekode);
2p        studentFeedbackDAO.registrereStudent(student);
1p        CookieUtil.leggCookieTilResponse(response, "nick", nick);
1p        response.sendRedirect("tilbakemelding");
    }
}
```

d) Skriv doGet()-metoden i GiTilbakemeldingServlet.java.

Forslag til løsning m/kommentarer og poenggiving (10p totalt er maks):

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

1p    String nick = CookieUtil.getCookieFraRequest(request, "nick");

1p    if (nick == null) { //Ingen cookie
1p        response.sendRedirect("registrerbruker");

    } else {
1p        Student student = studentFeedbackDAO.hentStudentMedNick(nick);

1p        if (student == null) { //Cookie, men ugyldig bruker
            response.sendRedirect("registrerbruker");

        } else {
1p            String datoOgTidNaa = TidOgDatoUtil.getNaaSomString();

1p            Aktivitet aktivitet = studentFeedbackDAO
                .hentAktuellAktivitetForStudent(student);

2p            request.setAttribute("datoOgTidNaa", datoOgTidNaa);
            request.setAttribute("student", student);
            request.setAttribute("aktivitet", aktivitet);

1p            request.getRequestDispatcher(
                "WEB-INF/Tilbakemeldingsskjema.jsp")
                .forward(request, response);

        }
    }
}
```

- e) Skriv enhetstest for NickValidator.java sin metode erGyldigNick(String nick). Ha et testutvalg som dekker grensetilfellene på en god måte.

Forslag til løsning m/kommentarer:

```
public class InputUtilTest {

    private NickValidator nickValidator = new NickValidator();

    @Test
    public void diverseGyldigInput() {

        assertTrue(nickValidator.erGyldigNick("abc"));
        assertTrue(nickValidator.erGyldigNick("20tegnlangt123456789"));
    }

    @Test
    public void diverseUgyldigInput() {

        assertFalse(nickValidator.erGyldigNick(null));
        assertFalse(nickValidator.erGyldigNick(""));
        assertFalse(nickValidator.erGyldigNick("ab"));
        assertFalse(nickValidator.erGyldigNick("21tegnlangt1234567890"));
        assertFalse(nickValidator.erGyldigNick("ab "));
        assertFalse(nickValidator.erGyldigNick("a c"));
        assertFalse(nickValidator.erGyldigNick("hallo verden"));
        assertFalse(nickValidator.erGyldigNick("20tegnplussmellomrom "));
    }
}
```

Merknad: Jeg burde også hatt med tester med æøå i nick, med store og små bokstaver, osv ... Det man tenker kan gå galt og det man tenker må være greit.

## Oppgave 5 (16% ~ 38 minutter) – Tråder

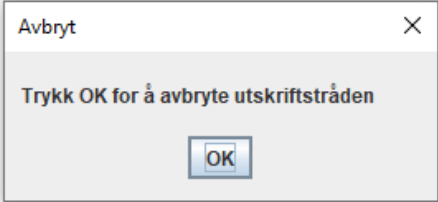
Du skal lage et lite program som starter opp to tråder i tillegg til main-tråden, avslutter de på kontrollert måte, og avslutter main når de to trådene er ferdige.

Strukturen til main() ser slik ut:

```
public static void main(String[] args) throws InterruptedException {  
  
    //Her opprettes og startes en tråd som går i en løkke og skriver ut  
    //en melding på skjermen ca. hvert sekund helt til tråden avsluttes  
    //på kontrollert måte.  
  
    //Her opprettes og startes en tråd som viser frem en JOptionPane meldings-  
    //boks. Når brukeren trykker på OK-knappen skal utskrifts-tråden få beskjed  
    //om å avslutte, og meldingsboks-tråden vil også være ferdig.  
  
    System.out.println("Tråder er startet. Venter på at de er ferdige ...");  
  
    //Her ventes det på at de andre trådene er ferdige før main avsluttes.  
  
    System.out.println("Begge tråder er ferdige!");  
    System.out.println("Main-tråd ferdig!");  
}
```

Kjøringen kan se ca. slik ut:

```
Tråder er startet. Venter på at de er ferdige ...  
Dette er en gjentakende melding !! :)  
Dette er en gjentakende melding !! :)  
Dette er en gjentakende melding !! :)  
Dette er en gjentakende melding !! :)  
Dette er en gjentakende melding !! :)  
Dette er en gjentakende melding !! :)
```



og etter at OK trykkes:

```
Begge tråder er ferdige!  
Main-tråd ferdig!
```

Meldingsboksen som venter på brukers OK kan fås frem slik:

```
JOptionPane.showMessageDialog(null, "Trykk OK for å avbryte utskriftstråden",  
    "Avbryt", JOptionPane.PLAIN_MESSAGE);
```

- a) Skriv koden for utskriftsloop-tråden. Du må selv avgjøre om dette kan/skal være en Thread eller en Runnable, og om den skal være konkret eller anonym.

Forslag til løsning m/kommentarer:

Jeg velger å lage en konkret subklasse av Thread.

Grunnen til at det ikke brukes Runnable (som kun har run()) er at vi også trenger en stopp()-metode. F.eks. slik:

```
public class PrintLoopTraad extends Thread {  
  
    private boolean fortsette = true;  
  
    public void stopp() {  
        fortsette = false;  
    }  
  
    @Override  
    public void run() {  
        while (fortsette) {  
            System.out.println("Dette er en gjentakende melding !! :)");  
            try {  
                sleep(1000);  
            } catch (InterruptedException e) {  
            }  
        }  
    }  
}
```

Merknad: De fleste av besvarelsene kommuniserer mellom trådene (stoppmelding) via en global boolsk variabel. Selv om dette «virker» er ikke dette noen god praksis. Objekter som «sender meldinger» til hverandre er mer på linje med objektorientert tankegang. Denne trådklassen bør derfor ha en stopp()-metode som brukes til å stoppe tråden!!

- b) Skriv koden for meldingsboks-tråden. Du må selv avgjøre om dette kan/skal være en Thread eller en Runnable, og om den skal være konkret eller anonym.

Forslag til løsning m/kommentarer:

Jeg velger også her å lage en konkret subklasse av Thread. Grunnen til at det ikke brukes Runnable (som kun har run()) er at vi også trenger en konstruktør som gir oss en referanse til den andre tråden. F.eks. slik:

```
public class AvbrytDialogTraad extends Thread {  
  
    private PrintLoopTraad printLoop;  
  
    public AvbrytDialogTraad(PrintLoopTraad printLoop) {  
        this.printLoop = printLoop;  
    }  
  
    @Override  
    public void run() {  
        JOptionPane.showMessageDialog(null,  
            "Trykk OK for å avbryte utskriftstråden",  
            "Avbryt", JOptionPane.PLAIN_MESSAGE);  
        printLoop.stopp();  
    }  
}
```

- c) Skriv koden i main som oppretter, starter og venter på disse trådene (slik det er vist i programkoden til main over.

Forslag til løsning m/kommentarer:

```
public static void main(String[] args) {  
  
    //Her opprettes og startes en tråd som går i en løkke og skriver ut  
    //en melding på skjermen ca. hvert sekund helt til tråden avsluttes  
    //på kontrollert måte.  
    PrintLoopTraad printLoop = new PrintLoopTraad();  
    printLoop.start();  
  
    //Her opprettes og startes en tråd som viser frem en JOptionPane meldings-  
    //boks. Når brukeren trykker på OK-knappen skal utskrifts-tråden få beskjed  
    //om å avslutte, og meldingsboks-tråden vil også være ferdig.  
    AvbrytDialogTraad avbrytDialog = new AvbrytDialogTraad(printLoop);  
    avbrytDialog.start();  
  
    System.out.println("Tråder er startet. Venter på at de er ferdige ...");  
    //Her ventes det på at de andre trådene er ferdige før main avsluttes.  
    printLoop.join();  
    avbrytDialog.join();  
  
    System.out.println("Begge tråder er ferdige!");  
    System.out.println("Main-tråd ferdig!");  
}
```

Merknad: En organisering av koden der all slags finurligheter legges i main er ikke god kode. Main bør være relativt ryddig!