

## **EKSAMENSOPPGAVE**

**Emnekode: DAT108**

**Emnenavn: Programmering og webapplikasjoner**

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

**Dato: 15. juni 2021**

---

Eksamensform: Skriftlig hjemmeeksamen (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

**NB! Det er ikke lov å ha kontakt med medstudenter eller andre personer under eksamen, eller å "dele" løsninger på oppgaver.**

**LYKKE TIL!**

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

For å få full score må løsningene ikke være unødig 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.
- 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.

- 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 hver tegn (J A V A).
- 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)

## 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?

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

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

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 *finnhyppestord*.

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 *finnhypigsteord*

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 *toLowerCase* 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 *finnhypigsteord* 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.

## 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.

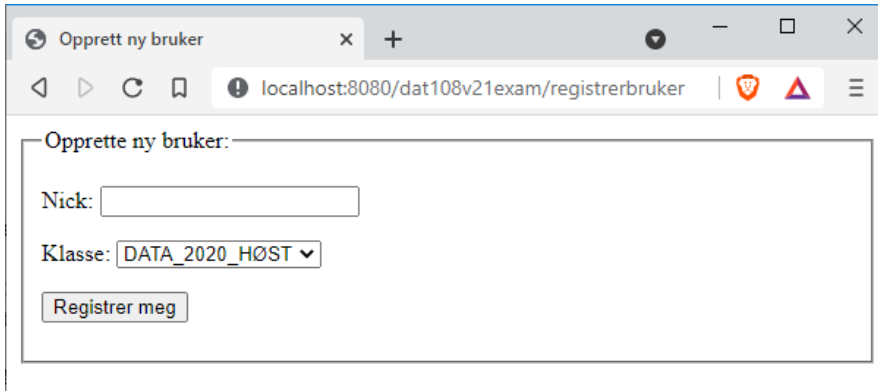
Det er gjort slik for at det skal vere enklere å bruke applikasjonen.

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 two input fields: 'Nick:' with an empty text box, and 'Klasse:' with a dropdown menu showing 'DATA\_2020\_HØST'. Below these fields is a button labeled 'Registrer meg'.

fig. 4.2

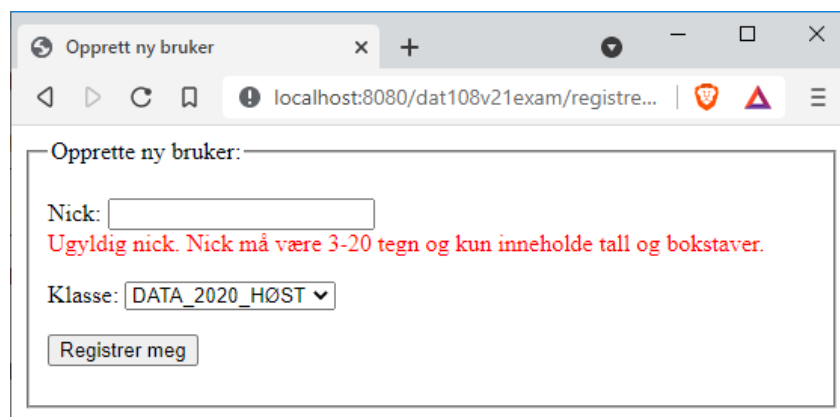
Rullegardin-listen av klassekoder er hentet fra en database (via **StudentFeedbackDAO**).

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 above the 'Klasse:' dropdown: 'Ugyldig nick. Nick må være 3-20 tegn og kun inneholde tall og bokstaver.' The 'Nick:' field is empty, and the 'Registrer meg' button is still visible.

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.
- b) Skriv RegistrerBrukerSkjema.jsp
- c) Skriv doPost()-metoden i RegistrerBrukerServlet.java.
- d) Skriv doGet()-metoden i GiTilbakemeldingServlet.java.
- e) Skriv enhetstest for NickValidator.java sin metode erGyldigNick(String nick). Ha et testutvalg som dekker grensetilfellene på en god måte.

## 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.

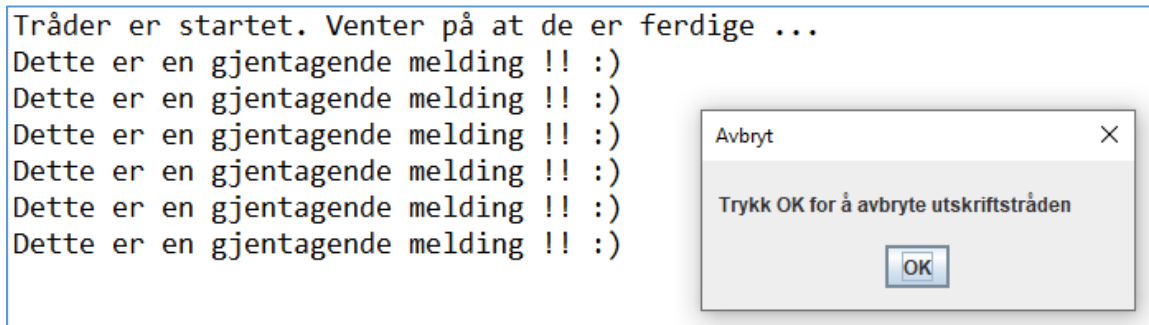
    //Her opprettes og startes en tråd som viser frem en JOptionPane meldings-
    //boks. Når brukeren trykker på OK-knappen skal utskrifts-tråden få beskjed
    //om å avslutte, og meldingsboks-tråden vil også være ferdig.

    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:



og etter at OK trykkes:



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

# **EKSAMENSOPPGÅVE**

**Emnekode: DAT108**

**Emnenamn: Programmering og webapplikasjoner**

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

**Dato: 15. juni 2021**

---

Eksamensform: Skriftleg hjemmeeksamen (Wiseflow | FLOWassign)

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

Tal på eksamensoppgaver: 5

Tal på sider (medrekna denne): 14

Tal på vedlegg: Ingen

Tillatne hjelpemiddel: Alle

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

**NB! Det er ikkje lov å ha kontakt med medstudentar eller andre personar under eksamen, eller å "dele" løsnings på oppgåver.**

**LUKKE TIL!**

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

For å få full score må løysingane ikkje vera unødige kompliserte.

Vi har definert eit interface med ein metode som tar inn ein streng og returnerer ein annan streng, t.d. som returnerer ei endra utgåve av den som kom inn:

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

Vi ønsker å laga ein metode for å skriva ut tekst formatert, slik:

```
/**
 * Metoden endrar på strengen som kjem inn v.hj.a. Stringfunksjon format,
 * og skriv 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 manglar i kodelistinga over.
- b) Skriv ein **main**-metode som bruker skrivUtFormatert() til å skriva ut ein tekst (som du vel sjølv) med

- i. Berre store bokstaver, eks. (Java) -> **JAVA**
- ii. Innramma, eks. (Java) -> **[Java]**
- iii. Med blank mellom kvart teikn -> **J a v a**

Bruk lambda-uttrykk til å representera Stringfunksjon-objekta, og tilordna dei helst til variablar med gode namn før dei blir brukte.

- c) Skriv ein metode **kombiner(...)** som tar inn to Stringfunksjon-ar og returnerer ein Stringfunksjon som gir ein kombinasjon av dei som blir putta inn. T.d. kan dette brukast til å gi ei utskrift som er ein kombi.. av store bokstavar og blanke mellom kvart teikn (J A V A).
- d) Anta nå at du har ei liste **spraakliste** som inneheld namn på ein del programmeringsspråk. Skriv kode som gir ei utskrift av disse språka, bortover på éi linje, innramma (ref. b.ii). Eks: **[Java][C#][Haskell][Rust]**. Dette skal gjerast på to ulike måtar:
  - i. Ved å bruka streams-APIet, men utan å bruke skrivUtFormatert()
  - ii. Ved å bruka skrivUtFormatert() (så enkelt som du klarer)



## Oppgave 2 (8% ~ 20 minutt) – Passordtryggleik

Når vi lagrar passord **salter**, **hasher** og **itererer** vi for å generera det som skal lagrast.

- a) Kva er hashing? Kvifor hasher vi i staden for å bruka vanleg kryptering?
- b) Kva er salting? Kvifor saltar vi?
- c) Kva er iterering (key stretching)? Kvifor itererer vi?

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

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

```
"use strict";
```

- i. Kva er konsekvensane av å leggje denne teksten i JavaScript-koden?
- ii. Kvar i JavaScript-koden skriv vi denne teksten?

b) Nøkkelordet *this* og mista kontekst.

- i. I nokre høve vil *this* miste konteksten av objektet som eig metoden som nyttar *this*. Gje døme på situasjonar når dette skjer, og forklar årsaka.
- ii. Gje døme på løysingar for å bevare konteksten ved bruk av *this*.
- iii. I nokre høve kan bruk av pil-syntaks for funksjonar løyse problem knytt til bruk av *this*. Vis med døme kva som er problemet og forklar korleis pil-syntaks kan løyse problemet.

c) Opprett ei JavaScript klasse **TekstAnalyse**. Klassen har følgjande metodar som kan blir nytta utanfor klassen (blir nytta **public**):

- Setter *tekst*. Du kan i staden nytte ein metode *setTekst* om du ikkje veit korleis du lagar ein setter.
- Getter *liste*. Du kan i staden nytte ein metode *getListe* om du ikkje veit korleis du lagar ein getter.
- Getter *antallord*. Du kan i staden nytte ein metode *getAntallord* om du ikkje veit korleis du lagar ein getter.
- Metode *finnhypigsteord*.

Nedanfor blir *getterne*, *setterne* og metodane omtala, med tips om korleis oppgåva kan bli løyst.

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

Denne registrerer ein tekst av ord som klassen skal arbeide med.

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

Denne returnerer ein **Array** førekomst av alle ord i teksten. Same ord kan førekome fleire gangar om det opptre fleire gangar i teksten.

Metoden *match* på førekomstar av **String** let oss enkelt fylla ein **Array** med ord frå ein tekst. Metoden tar eit regulært uttrykk som argument og returnerer ein **Array** av dei delane av teksten som passer med mønsteret. Eit uttrykk som kan bli nytta for denne oppgåva er:

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

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

Denne returnerer tal på ord i teksten. Same ord blir tald fleire gangar om det blir funnen fleire gangar i teksten.

### Metode *finnhypigsteord*

Denne metoden skal returnere dei orda som blir funnen flest gangar i teksten, og tal på gangar desse orda blir funnen.

Observer at forskjellige ord kan bli funnen like mange gangar, så derfor må metoden returnere ein **Array** førekomst av ord. Returnerte skal være følgjande objekt:

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

Her er *maksantall* det største tal på gangar eit ord finnast i teksten, og *ordliste* er ein **Array** av ord.

Metoden skal ikkje skilja på store og små bokstaver når ord blir talde.

For å finne tal på førekomstar av kvart einskild ord kan du bruke ein **Map**, der nøkkel er ordet, og verdi er tal på førekomstar av ordet. Ein **Map** sin metode *has* kan sjekke om ein nøkkel finns, og metodane *get* og *set* blir nytta for å hente ut og leggje til eit element til ein førekomst **av Map**.

For å unngå å skilje på store og små bokstaver må orda bli konvertert når dei blir legga inn i ein **Map**, til dømes ved å nytte metoden *toLocaleUpperCase* som kan bli nytta på førekomstar av **String**.

### Kode døme

Døme under opprettar ein **TekstAnalyse** og legg inn ein tekst:

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

Nedanfor kan du sjå korleis getteren *antallord* blir nytta:

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

Under kan du sjå utskrifta i nettlesarkonsollet frå koden over:

```
Tal på ord i teksten er 5
```

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

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

Metoden *finnhypigsteord* skal returnere følgjande **Object** for den gitte teksten:

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

Med *ordmap* ein **Map** av ord kan vi legge til eit nytt ord frå variabel *nyttord* med følgjande kode:

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

I koden ovanfor er verdi satt til 1 då dette var eit nytt ord, og ordet nå er registrert ein gang.

### Oppgåve

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

## Oppgave 4 (40% ~ 96 minutt) – Webapplikasjonar, tenerside

### Innleiing

Etter ei forelesning lurar ofte forelesar på korleis det har gått. Synest studentane forelesningen var god? Var det noko som var uklart? Kva kunne vore gjort annleis? Studenten har kanskje tilsvarande behov: Å gi tilbakemelding, stilla spørsmål, seia frå om noko som var uklart.

Vi ønsker derfor å laga ein web-applikasjon der studentar enkelt kan gi tilbakemelding etter ein forelesning (eller ein lab-time). Sidan for tilbakemelding kan til dømes sjå slik ut:



fig. 4.1

**NB! Det skal ikkje vera innlogging i tradisjonell forstand.**

Ein (student) brukar blir identifisert ved at ein persistent cookie er lagra i nettlesaren. Denne blir lagra ved registrering (sjå neste side), og cookien inneheld brukaren sin nick.

Det er gjort slik for at det skal vere enklare å bruka applikasjonen.

Vi skal ikkje sjå på heile applikasjonen i denne eksamensoppgåva, berre eit par utvalde sider og brukstilfelle.

### Brukstilfelle – Oppretta ny bruker (for ein student)

Ein må registrera seg som bruker for å kunna gi tilbakemelding. Dette blir gjort for å kopla student til klasse slik at rett aktivitet (frå timeplanen) kjem fram til rett tid..

Skjema for registrering kan til dømes sjå slik ut:

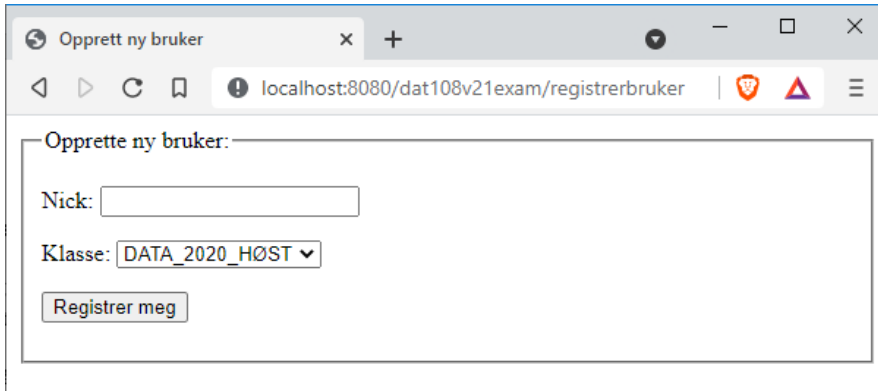
A screenshot of a web browser window titled 'Opprett ny bruker'. The address bar shows 'localhost:8080/dat108v21exam/registrerbruker'. The form contains a 'Nick:' text input field, a 'Klasse:' dropdown menu with 'DATA\_2020\_HØST' selected, and a 'Registrer meg' button.

fig. 4.2

Rullegardin-listen av klassekodar er henta frå ein database (via **StudentFeedbackDAO**).

Når ein registrerer seg (og alt går fint), blir student (nick + klassekode) lagra i databasen (via **StudentFeedbackDAO**). Nick-et blir òg sendt til nettlesar som en persistent cookie (via **CookieUtil**). Ved seinare bruk (fra same nettleser) blir bruker automatisk identifisert via denne cookien.

Viss registreringa går fint, blir bruker sendt til tilbakemeldingssida (fig. 4.1).

Feilsituasjoner ved registrering:

- Ved oppretting av bruker skal **nick** validerast. Eit nick må vera 3-20 teikn langt og kan berre innehalda tal og bokstavar. Ved ugyldig nick skal brukar sendast tilbake til same side, og ei feilmelding skal visast (fig. 4.3).
- Ved oppretting av bruker kan **nick** ikke allereie eksistere i databasen. Ved allereie eksisterande nick skal brukar sendast tilbake til same side, og ei anna feilmelding skal visast.

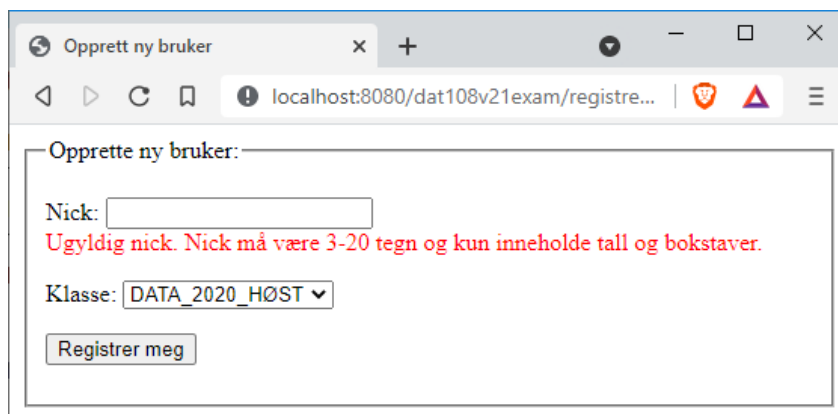
A screenshot of the same web browser window as in fig. 4.2, but now displaying a red error message below the 'Nick:' input field: 'Ugyldig nick. Nick må være 3-20 tegn og kun inneholde tall og bokstaver.' The 'Klasse:' dropdown and 'Registrer meg' button remain visible.

fig. 4.3

## Brukstilfelle – Gi tilbakemelding

Frå hovudsida (fig. 4.1) kan ein brukar gi tilbakemelding på ein aktivitet for klassen sin.

Tekst i utheva skrift på sida (fig. 4.1) er dynamiske data.

- Pent formatert dato og klokkeslett kjem frå en **TidOgDatoUtil**.
- Studenten sitt nick ("lph") kjem frå nettleser-cookie, som kan hentast via **CookieUtil**, men må òg finnast i databasen, representert ved **StudentFeedbackDAO**.
- Klassekode er kopla til nick, og blir henta frå databasen via **StudentFeedbackDAO**.
- Aktuell aktivitet (se fig. 4.1) kjem frå **StudentFeedbackDAO**, og er berekna ut frå gjeldande tidspunkt og timeplan for aktuell klasse.

Feilsituasjoner:

- Viss ein prøver å nå denne sida utan at ein har ein cookie med eit gyldig og registrert nick, skal ein omdirigerast til registrerings skjemaet. (Då må ein registrera seg på nytt. NB! Ein kan ikkje registrera seg med same nick fleire gangar / i fleire nettlesarar.)

Ved trykking på Send-knappen i fig. 4.1:

- **Kva som skjer ved sending av tilbakemelding er ikkje ein del av denne eksamensoppgåva.**

**Klasser og jsp-er i applikasjonen:****@Entity**

**Student.java** representerer ein studentbrukar. Har følgjande eigenskaper:

- @Id String nick;
- String klassekode;

**@Entity**

**Aktivitet.java** representerer ein aktivitet. Har følgjande eigenskaper:

- ... (eigenskapene er ikkje naudsynte for eksamensoppgåva)

og følgjande metode:

- String getVisningstekst() // Denne kan brukast i JSP-sida som tekst for gjeldande aktivitet

**@Stateless****StudentFeedbackDAO.java**

StudentFeedbackDAO er ein stateless session bean, og skal derfor brukast som ein @EJB i servletane. StudentFeedbackDAO bruker JPA til å kommunisere med den underliggjande 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 brukast i JSP-sida 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 legg til responsen

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

**NickValidator.java**

// Et nick må vera 3-20 teikn langt og kan berre innehalde tal og bokstavar.

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



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

- **GET** – Ein førespurnad om å få sjå sida for brukarregistrering, ref. fig. 4.2. Samarbeid med viewet **RegistrerBrukerSkjema.jsp**. Viss sida skal innehalda ei feilmelding er type feilmelding gitt som ein parameter **feilkode** til førespurnaden: 1=ugyldig nick, 2=nick finst frå før.
- **POST** – Ein førespurnad om å registrera ein ny brukar. Viss alt går bra skal ein til slutt omdirigerast til **tilbakemelding**. Viss ikkje alt går bra, skal ein omdirigerast til **registrerbruker** igjen.

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

- **GET** - Ein førespurnad om å få sjå sida for tilbakemeldingsskjema, ref. fig. 4.1. Samarbeid med viewet **Tilbakemeldingsskjema.jsp**. Alternativt scenario:
  - Viss ein ikkje har ein cookie med eit gyldig nick for ein registrert brukar, skal ein omdirigerast til **registrerbruker**.
- **POST** - Ein førespurnad om å registrera ei tilbakemelding. **Ikkje ein del av eksamensoppgåva.**

### **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. **Ikkje ein del av eksamensoppgåva.**

### **Krav til løysinga**

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

Løysinga treng ikkje å vera robust mot «hacking», dvs. utilsikta bruk.

Løysinga treng ikkje å vera robust mot tekniske databasefeil.

**No til sjølve oppgåvene**

- a) Skriv doGet()-metoden i RegistrerBrukerServlet.java.
- b) Skriv RegistrerBrukerSkjema.jsp
- c) Skriv doPost()-metoden i RegistrerBrukerServlet.java.
- d) Skriv doGet()-metoden i GiTilbakemeldingServlet.java.
- e) Skriv einingstest (enhetstest) for NickValidator.java sin metode erGyldigNick(String nick). Ha eit testutval som dekker grensetilfella på ein god måte.

## Oppgave 5 (16% ~ 38 minutt) – Trådar

Du skal laga eit lite program som startar opp to trådar i tillegg til main-tråden, avsluttar dei på kontrollert måte, og avsluttar main når dei to trådane er ferdige.

Strukturen til main() ser slik ut:

```
public static void main(String[] args) throws InterruptedException {

    //Her opprettes og startes ein tråd som går i en løkke og skriver ut
    //ei melding på skjermen ca. kvart sekund heilt til tråden blir avslutta
    //på kontrollert måte.

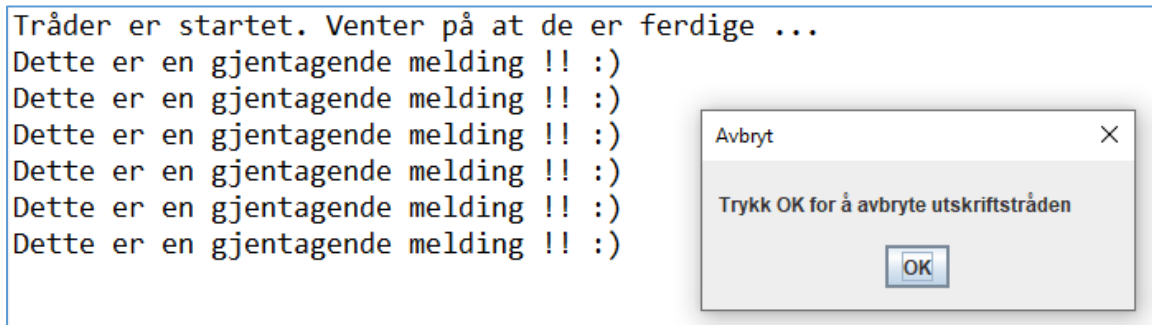
    //Her opprettes og startes ein tråd som viser frem ein JOptionPane meldings-
    //boks. Når brukaren trykker på OK-knappen skal utskrifts-tråden få beskjed
    //om å avslutta, og meldingsboks-tråden vil òg vera ferdig.

    System.out.println("Tråder er startet. Venter på at de er ferdige ...");

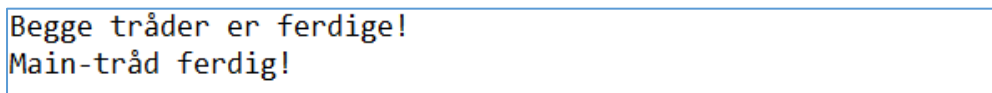
    //Her ventes det på at dei andre trådane er ferdige før main blir avslutta.

    System.out.println("Begge tråder er ferdige!");
    System.out.println("Main-tråd ferdig!");
}
```

Køyringa kan sjå ca. slik ut:



og etter at OK blir trykt:



Meldingsboksen som venter på brukars OK kan fåast fram slik:

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

- a) Skriv koden for utskriftsloop-tråden. Du må sjølv avgjera om dette kan/skal vera ein Thread eller ein Runnable, og om han skal vera konkret eller anonym.
- b) Skriv koden for meldingsboks-tråden. Du må sjølv avgjera om dette kan/skal vera ein Thread eller ein Runnable, og om han skal vera konkret eller anonym.
- c) Skriv koden i main som opprettar, startar og ventar på desse trådene (slik det er vist i programkoden til main over).