

EKSAMENSOPPGAVE

Sammensatt av flervalgs- (Wiseflow) og tekstoppgavene (pdf)

Emnekode: DAT108

Emnenavn: Programmering og webapplikasjoner

Klasse: 2. klasse DATA / INF

Dato: 18. desember 2023

Eksamensform: Skriftlig skoleeksamen (Wiseflow | FLOWmulti)

Eksamenstid: 4 timer (0900-1300)

Antall eksamensoppgaver: 6

Antall sider (medregnet denne): ---

Antall vedlegg: Ingen

Tillatte hjelpemiddel: Ingen

Lærere: Lars-Petter Helland (928 28 046) lph@hvl.no
 Erlend Raa Vågset (57 72 26 08) erlend.raa.vagset@hvl.no
 Bjarte Kileng (909 97 348) bki@hvl.no

LYKKE TIL!

Oppgave 1 (12%) – Strømmer

Flervalgsoppgave: Automatisk retting

Om oppgave A, B, C, D, E og F:

I hver deloppgave skal du lage en strøm (stream) som løser et problem gitt i oppgave teksten.

Oppgave A: Gitt en liste av ord (`List<String> ordInn`), lag en ny liste (`ordUt`) som kun inneholder ord som har odd lengde.

```
1  ordUt = ordInn. 2  ()  
   . 3  ( 4  )  
   . 5  ( );
```

Oppgave B: Gitt en liste med gaver (`List<Gave> onskeliste`), finn ut om alle de 10 billigste gavene er myke.

```
1  erMyke = onskeliste. 2  ()  
   . 3  ( 4  )  
   . 5  ( 6  )  
   . 7  ( 8  );
```

Om klassen Gave:

Klassen `Gave` har følgende feltvariablene (med tilsvarende `get`'ere og `set`'ere):

- `String navn;` // Navnet på gaven (For eksempel "Byggeklusser", "Genser" osv...)
- `Integer pris;` // Pris (i NOK)
- `Integer vekt;` // Vekt (i gram)
- `Boolean myk;` // Sann om gaven er myk, usann om gaven er hard

Oppgave C: Gitt en liste med gaver (List<Gave> ønskeliste), blandt de 20 billigste gavene, finn en hard gave blandt de 10 tyngste gavene (dersom en slik gave eksisterer).

<div style="border: 1px solid black; padding: 2px; display: inline-block;">1</div>	hardGave = ønskeliste.	<div style="border: 1px solid black; padding: 2px; display: inline-block;">2</div>	() . 3 . 5 . 7 . 9 . 11 . 13	(4) (6) (8) (10) (12) () ;
--	------------------------	--	---	--

Oppgave D: Gitt en liste av ønskelister (List<List<Gave>> innOnskelister), skriv ut navnene på alle de myke gavene fra de ønskelistene som har færre enn 10 gaver.

innOnskelister.	<div style="border: 1px solid black; padding: 2px; display: inline-block;">1</div> . <div style="border: 1px solid black; padding: 2px; display: inline-block;">2</div> . <div style="border: 1px solid black; padding: 2px; display: inline-block;">4</div> . <div style="border: 1px solid black; padding: 2px; display: inline-block;">6</div> . <div style="border: 1px solid black; padding: 2px; display: inline-block;">8</div>	() (<div style="border: 1px solid black; padding: 2px; display: inline-block;">3</div>) (<div style="border: 1px solid black; padding: 2px; display: inline-block;">5</div>) (<div style="border: 1px solid black; padding: 2px; display: inline-block;">7</div>) (<div style="border: 1px solid black; padding: 2px; display: inline-block;">9</div>) ;
-----------------	--	---

Oppgave E: Gitt en liste med ønskelister (List<List<Gave>> ønskelisteliste), finn snittprisen på gavene i den lengste ønskelisten. Hvis du får en tom liste av ønskelister eller hvis den lengste ønskelisten er tom så skal du returnere -1.

```

1  snittpris = ønskelisteliste. 2  ( )
    . 3  ( 4  )
    .orElse ( new ArrayList<Gave> ( )
);
    . 5  ( )
    . 6  ( 7  )
    . 8  ( 9  )
    . 10  ( )
    .orElse ( -1 );

```

Oppgave F: Lag et akronym/en forkortelse fra en liste med strenger (se eksempelet under).

```

1  akronym = strenger. 2  ( )
    . 3  ( 4  );

```

Eksempel:

- ["United", "States of", "America"] blir til "U.S.A."
- ["American", "Standard", "Code for", "Information", "Interchange"] blir til "A.S.C.I.I."
- ["light", "amplification by", "", "stimulated", "", "emission of", "radiation"] blir til "l.a.s.e.r." (Tomme strenger i listen fjernes).

Oppgave 2 (8%) – Funksjonelle kontrakter

Flervalgsoppgave: Automatisk retting

Om oppgave A, B, C og D:

I hver oppgave skal du bruke nedtrekksmenyen til å lage *gyldige* lambdauttrykk med *rett type* som *løser* problemet som oppgaveteksten beskriver.

Oppgave A: Gi rett type til variablene a1, a2 og a3 under:

// i)

1 2 a1 = (x, y) -> x + x*y + y;

// ii)

3 4 a2 = String::length;

// iii)

5 6 a3 = x -> x.toUpperCase().length();

Oppgave B: Velg det lambdauttrykket passer med typen til variabelene b1, b2 og b3.

```
// i)
Predicate<String> b1 =  ;

// ii)
Function<Integer, String> b2 =  ;

// iii)
Comparator<String> b3 =  ;
```

Oppgave C: Hvilket funksjonelle grensesnitt har lambdauttrykkene c1, c2, c3, c4 og c5, gitt at...

```
// i)
// Når vi skriver c1.apply("Kari") så returneres et objekt av typen Person.
  c1;

// ii)
// Når vi skriver c2.apply("Kari", 20) så returneres et objekt av typen Person.
  c2;

// iii)
// Når vi skriver c3.test(new Person("Kari", 20)) så returneres verdien true.
  c3;
```

Oppgave D: Velg rett lambdauttrykk og type til variablene d1, d2 og d3.

```
// i)
// Inn: En person (Person).
// Ut: Etternavnet til personen.
```

1 2 d1 = 3 ;

```
// ii)
// Inn: To personer (Person).
// Ut: Aldersforskjellen (differansen) mellom personene (Hint: Metoden skal brukes i
en strøm til sortering av en liste av Personer).
```

4 5 d2 = 6 ;

```
// iii)
// Inn: En person (Person) og et lambdauttrykk som regner ut inntektsskatt for en pe
rson.
// Ut: true dersom personen betaler mindre enn 500 000 NOK i skatt, ellers false.
```

7 8 d3 = 9 ;

Oppgave 3 (15% ~ 36 minutter) – Tråder

Om oppgaven:

Denne oppgaven består av fire deloppgaver, A, B, C og D. Oppgaven rettes manuelt og hver av deloppgavene teller omtrent like mye.

Koden under skulle simulere tre ulike typer nisser som arbeider *parallelt* på et juleverksted:

- Lagenisse, som lager leker (2 stk.)
- Pakkenisse, som pakker inn leker (3 stk.)
- Julenisse, som leverer pakker (1 stk.)

PROBLEMET ER AT KODEN UNDER IKKE OPPFØRER SEG SOM VI HADDE TENKT.

Målet med denne oppgaven er at du skal finne ut av hva det er som har gått galt og at du skal foreslå en løsning.

Ønsket oppførsel

Krav til programmet:

- Alle lekene skal ha et unikt nummer.
- Alle pakkene skal ha et unikt nummer.
- Alle lekene skal pakkes inn nøyaktig en gang.
- Alle pakkene skal leveres nøyaktig en gang.
- Leker skal ikke pakkes inn før de er lagd.
- Pakker skal ikke leveres før de er pakket inn.
- Når det er tomt for leker skal pakkenissen vente på neste leke.
- Når det er tomt for pakker skal julenissen vente på neste pakke.
- Når det ikke er plass til flere leker skal lagenissen vente med å legge fra seg leken til en pakkenisse har fjernet en leke.
- Når det ikke er plass til flere pakker skal pakkenissen vente med å legge fra seg pakken til julenissen har fjernet en pakke.
- Utover dette skal nissene *ikke* vente på hverandre.

Vi tillater at:

- Lekene ikke blir skrevet ut i sortert rekkefølge.
- Pakkene ikke blir skrevet ut i sortert rekkefølge.
- En leke har samme nummer som en pakke.
- En pakke inneholder en leke med et annet nummer enn det pakken har.

Eksempel på godkjent utskrift:

Leke 1

Leke 2

Leke 4

Leke 3

Pakke 1 [Leke 2]

Pakke 2 [Leke 1] osv...

Eksempel på feil i utskrift:

Leke 1

Leke 2

Leke 3

Leke 3

Pakke 1 [Leke 2]

Pakke 1 [Leke 1] osv...

(Det er nøyaktig to feil her: 1) at "Leke 3" er skrevet ut to ganger og 2) at "Pakke 1" er skrevet to ganger)

KODE

Klassen Juleverksted:

```
1 | import java.util.concurrent.*;
2 |
3 | class Juleverksted {
4 |     private final BlockingQueue<String> leker;
5 |     private final BlockingQueue<String> pakker;
6 |     private int lekeTeller = 0;
7 |     private int pakkeTeller = 0;
8 |
9 |     Juleverksted(int kapasitet) {
10 |         leker = new ArrayBlockingQueue<>(kapasitet);
11 |         pakker = new ArrayBlockingQueue<>(kapasitet);
12 |     }
13 |
14 |     // Metode som simulerer at en nisse lager en leke.
15 |     public void lagLeke() throws InterruptedException {
16 |         Thread.sleep(2000); // Simul
17 |         ++lekeTeller;
18 |         String leke = "Leke " + lekeTeller;
19 |         System.out.println(leke);
20 |         leker.put(leke);
21 |     }
22 |
23 |     // Metode som simulerer at en nisse pakker inn en leke.
24 |     public void pakkInnLeke() throws InterruptedException {
25 |         String leke = leker.take(); // Simul
26 |         Thread.sleep(3000);
27 |         ++pakkeTeller;
28 |         String pakke = "Pakke " + pakkeTeller + " [" + leke + "]";
29 |         System.out.println(pakke);
30 |         pakker.put(pakke);
31 |     }
32 |
33 |     // Metode som simulerer at en nisse leverer en pakke.
34 |     public void leverPakke() throws InterruptedException {
35 |         String pakke = pakker.take(); // Simul
36 |         Thread.sleep(1000);
37 |         System.out.println("Leverer : " + pakke);
38 |     }
39 | }
40 |
```

Klassen Lagenisse:

```
41 | class Lagenisse implements Runnable {
42 |     private final Juleverksted juleverksted;
43 |
44 |     Lagenisse(Juleverksted juleverksted) {
45 |         this.juleverksted = juleverksted;
46 |     }
```

```
47 |
48 |     @Override
49 |     public void run() {
50 |         while (true) {
51 |             try {
52 |                 juleverksted.lagLeke();
53 |             } catch (InterruptedException e) {
54 |
55 |             }
56 |         }
57 |     }
58 | }
59 |
```

Klassen Pakkenisse:

```
60 | class Pakkenisse implements Runnable {
61 |     private final Juleverksted juleverksted;
62 |
63 |     Pakkenisse(Juleverksted juleverksted) {
64 |         this.juleverksted = juleverksted;
65 |     }
66 |
67 |     @Override
68 |     public void run() {
69 |         while (true) {
70 |             try {
71 |                 juleverksted.pakkInnLeke();
72 |             } catch (InterruptedException e) {
73 |
74 |             }
75 |         }
76 |     }
77 | }
78 |
```

Klassen Julenisse:

```
79 | class Julenisse implements Runnable {
80 |     private final Juleverksted juleverksted;
81 |
82 |     Julenisse(Juleverksted juleverksted) {
83 |         this.juleverksted = juleverksted;
84 |     }
85 |
86 |     @Override
87 |     public void run() {
88 |         while (true) {
89 |             try {
90 |                 juleverksted.leverPakke();
91 |             } catch (InterruptedException e) {
92 |
93 |             }
94 |         }
```

```
95 |     }
96 | }
97 |
```

Klassen JuleverkstedEksempel (main metode):

```
98 | // Main metoden
99 | public class JuleverkstedEksempel {
100 |     public static void main(String[] args) {
101 |         Juleverksted juleverksted = new Juleverksted(10); // Maksimal kapasite
t på verkstedet
102 |
103 |         // Oppretter 1 Julenisse, 2 Lagenisser og 3 Pakkenisser
104 |         Thread julenissen = new Thread(new Julenisse(juleverksted));
105 |         Thread lagenisse1 = new Thread(new Lagenisse(juleverksted));
106 |         Thread lagenisse2 = new Thread(new Lagenisse(juleverksted));
107 |         Thread pakkenisse1 = new Thread(new Pakkenisse(juleverksted));
108 |         Thread pakkenisse2 = new Thread(new Pakkenisse(juleverksted));
109 |         Thread pakkenisse3 = new Thread(new Pakkenisse(juleverksted));
110 |
111 |         julenissen.start();
112 |         lagenisse1.start();
113 |         lagenisse2.start();
114 |         pakkenisse1.start();
115 |         pakkenisse2.start();
116 |         pakkenisse3.start();
117 |     }
118 | }
```

Oppgave A:

Pek på de linjene i koden hvor du finner kappløpstilstander (race conditions) som du mener er problematiske. Forklar hva det er som kan gå galt.

Oppgave B:

Hvorfor er det en dårlig ide å fikse problemene vi har ved å bruke synkronisering på metodene i koden over?

Oppgave C:

Hvordan kunne vi gjort koden over trådsikker? Velg deg en metodene over som ikke er trådsikker og gjør den trådsikker (uten å over-synkronisere).

Oppgave D:

På hvilke kodelinjer ville vi fått problemer dersom vi hadde brukt en Queue istedenfor en BlockingQueue? Forklar hva det er som kan gå galt.

Oppgave 4 (10% ~ 24 minutter) – JavaScript teori

Flervalgsoppgave: Automatisk retting

I flervalgsoppgavene kan ingen, ett eller flere av utsagnene være korrekte. En flervalgsoppgave gir ingen poeng hvis et av de feile utsagnene er valgt, selv om også et riktig valg er gjort. Helgartering gir kun poeng hvis alle utsagnene er korrekte.

a) En webside skal kjøre JavaScript-koden under:

```
window.alert('Velkommen');
```

Hvilke av alternativene under kan brukes for å få utført-koden?

1. Lagre koden i en fil *index.js*. En webside *index.html* vil alltid laste JavaScript-kode fra en fil *index.js*.
2. Lagre koden i en fil *index.js*. Legg så inn følgende kode i HTML-koden:

```
<script src="index.js"></script>
```

3. Legg inn i HTML-koden:

```
<code>window.alert('Velkommen')</code>
```

4. Legg inn i HTML-koden:

```
<script>window.alert('Velkommen')</script>
```

5. Legg inn i HTML-koden:

```
<script javascript="window.alert('Velkommen')"></script>
```

6. Lagre koden i en fil *index.js* og legg inn i HTML-koden:

```
<javascript>index.js</javascript>
```

b) Hvilke JavaScript-objekt kan gi oss operativsystemet til klienten?

1. window,
2. navigator,
3. history,
4. screen,
5. document.

c) En webside skal kjøre JavaScript-koden under:

```
const liListe = document.getElementsByTagName("li");
while (liListe.length > 0) {
    liListe[0].remove();
}
```

Hvilke av utsagnene under er korrekt(e)?

1. Koden forsøker å fjerne samme HTML LI element flere ganger.
2. Koden vil feile da argumentet «li» til *getElementsByTagName* ikke er en lovlig CSS-selektor.
3. Nettleser vil oppdatere hvilket element i listen som er første element og lengden av listen fortløpende, og løkken vil stoppe når siste element er fjernet.
4. Løkken vil aldri stoppe da *liListe.length* ikke blir oppdatert.
5. Koden vil feile siden tagg-navn må skrives med store bokstaver, dvs. *LI* for tagg .
6. JavaScript kan legge til HTML elementer, men ikke fjerne element. Koden vil derfor feile.

d) En webside skal kjøre JavaScript-koden under:

```
const liListe = document.querySelectorAll("li");
while (liListe.length > 0) {
    liListe[0].remove();
}
```

Hvilke av utsagnene under er korrekt(e)?

1. Koden forsøker å fjerne samme HTML LI element flere ganger.
2. Koden vil feile da argumentet «li» til *querySelectorAll* ikke er en lovlig CSS-selektor.
3. Nettleser vil oppdatere hvilket element i listen som er første element og lengden av listen fortløpende, og løkken vil stoppe når siste element er fjernet.
4. Løkken vil aldri stoppe da *liListe.length* ikke blir oppdatert.
5. Koden vil feile siden tagg-navn må skrives med store bokstaver, dvs. *LI* For tagg .
6. JavaScript kan legge til HTML elementer, men ikke fjerne element. Koden vil derfor feile.

e) Hva gjør koden under?

```
let [a,b] = data;
```

1. Koden er ulovlig da vi ikke kan bruke JSON syntaks for **Array** på venstre side av et likhetstegn.
2. Koden returnerer **true** hvis begge sider av likhetstegnet er en **Array** med likt innhold.
3. Koden tilordner variabel *a* til verdien til *data*, mens *b* blir *undefined*.
4. Variabel *data* må være en liste-struktur med nøyaktig to elementer.
5. Variabel *data* må være en liste-struktur minst to elementer.
6. Koden vil tilordne verdier til variablene *a* og *b* fra elementer i *data* som må være en liste-struktur.

f) Koden under er gitt:

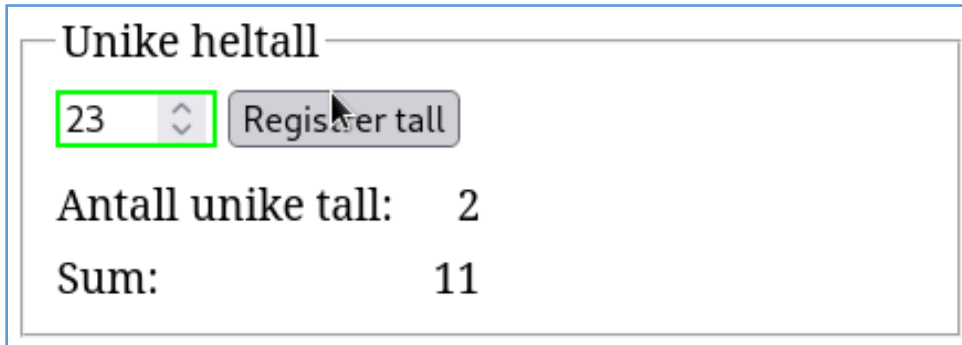
```
summer(...[1,4,2,8]);
```

Hvilke av utsagnene under er korrekt(e)?

1. Konstruksjonen med tre punktum «...» er ulovlig.
2. Antall punktum angir hvor mange elementer som skal hentes ut fra listen med tall. Funksjonen *summer* kjøres derfor med argumentene 1, 4 og 2.
3. Funksjonen *summer* kjøres med 4 argumenter der de tre første er tegnet punktum «.», og siste argument er en liste av tall.
4. Funksjonen *summer* kjøres med 4 argumenter som er 1, 4, 2, og 8.
5. Funksjonen *summer* kjøres med 4 argumenter der de tre første er *undefined* og siste er en liste av tall.
6. Funksjonen *summer* kjøres med 4 argumenter der de tre første er *null* og siste er en liste med tall.

Oppgave 5 (15% ~ 36 minutter) – JavaScript programmering

En webside inneholder et felt som lar bruker registrere unike heltall, og som viser antallet med unike tall og summen av dem. Skjermbildet under viser hvordan websiden kan se ut:



Skjermbilde 1: Felt for å registrere heltall

I skjermbildet over har bruker allerede registrert tallene 4 og 7, og holder nå på med å registrere tallet 23.

Applikasjonen bruker HTML-koden under for å lese inn tallene:

```
<fieldset id="root">
  <legend>Unike heltall</legend>

  <div>
    <input type="number" size="5">
    <button type="button">Registrer tall</button>
  </div>
  <div class="resultat">
    Antall unike tall: <span>0</span>
    Sum: <span>0</span>
  </div>
</fieldset>
```

Kode-eksempel 1: HTML-kode for å lese inn heltall

Du kan anta at applikasjonen kun vil lese heltall, da alt annet vil sette input-feltet sin tilstand til *invalid*. Dette gjøres automatisk av nettleser. Når feltet sin tilstand er *invalid* endres rammefargen til rød. Dette gjøres av CSS-kode. Dere skal ikke skrive CSS i denne oppgaven!

Input-feltet sin tilstand skal settes til *invalid* også hvis bruker forsøker å registrere et tall som allerede er registrert. Dette må håndteres av JavaScript-koden. Dette er illustrert i skjermbildet under. Observer at dere ikke skal skrive CSS-koden som gjør rammen rød, kun settet input-feltet sin tilstand til *invalid*.



Skjerm bilde 2: Forsøk på å registrere et tall som allerede er registrert

Applikasjonen inneholder følgende JavaScript-kode:

```
class Talldata {
  #tallinput;

  // Legg til her eventuelle flere private felt

  constructor(root) {
    this.#tallinput =
      root.getElementsByTagName("input")[0];
    const bt =
      root.getElementsByTagName("button")[0];

    bt.addEventListener("click",
      (event) => { this.#lestall(event) }
    );

    // Legg inn her mer kode hvis nødvendig
  }

  #lestall(event) {
    // Legg inn kode her
  }

  // Legg til eventuelle flere private metoder
}

const rootelement = document.getElementById("root");
new Talldata(rootelement);
```

Oppgave: Fyll inn den manglende koden i JavaScript-klassen **Talldata**.

Hjelp:

- Kode-eksempelet under setter tilstanden til *invalid* for et input element og viser feilmeldingen «Noe er feil»:

```
const input = document.getElementsByTagName("input")[0];
input.setCustomValidity("Noe er feil");
input.reportValidity();
```

- Kode-eksempelet under setter tilstanden til *valid* for et input element og fjerner feilmeldingen:

```
const input = document.getElementsByTagName("input")[0];
input.setCustomValidity("");
input.reportValidity();
```

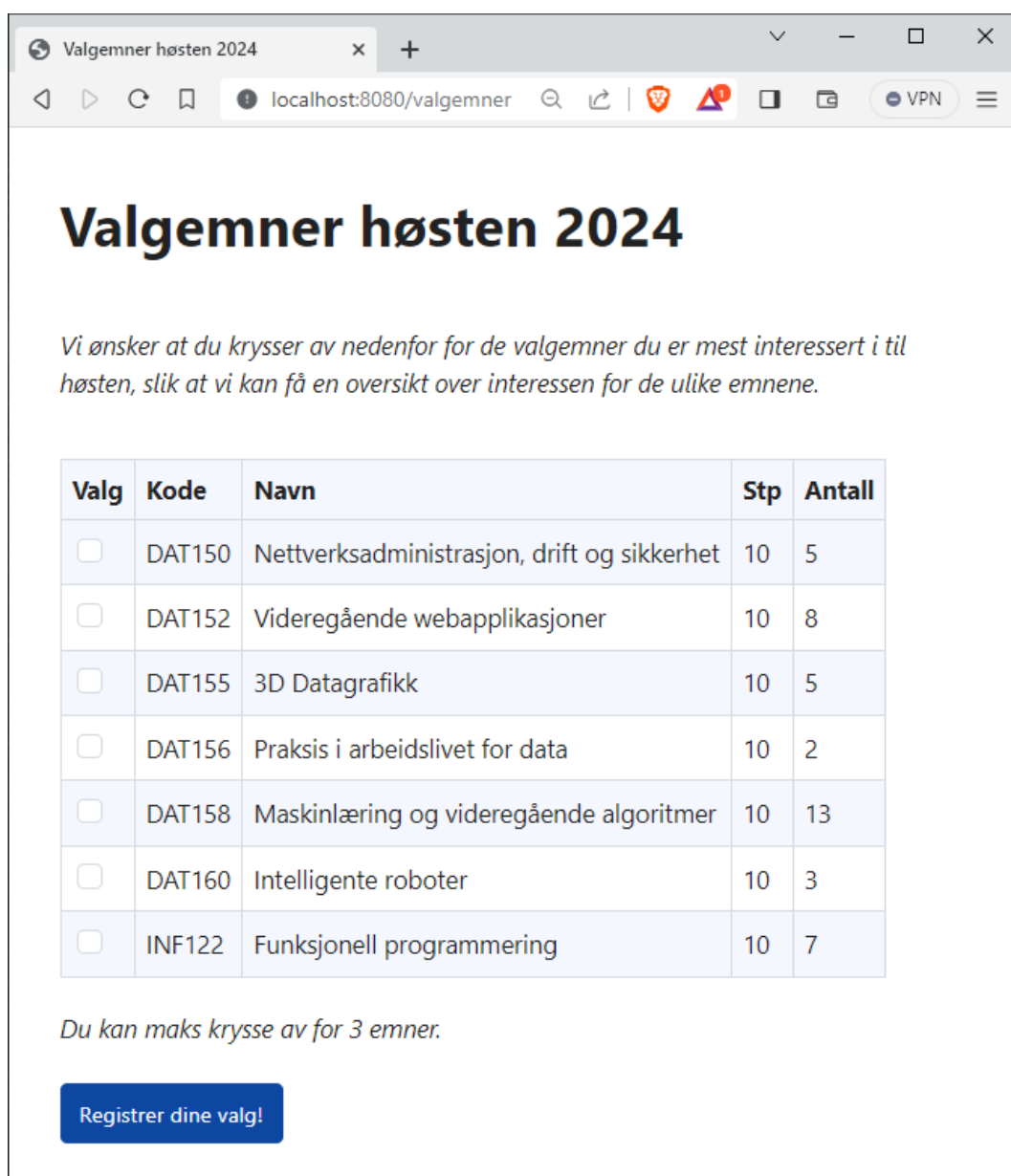
- Både **Map**, **Set** og **Array** har en metode *forEach*.
- **Map** og **Set** sin metode *values* returnerer en iterator over alle verdier.
- **Map** sin metode *keys* returnerer en iterator over alle nøkler.
- **Array** sin statiske metode *from*, og også spre (*spread*) operator kan kopiere et itererbart objekt til en **Array**.
- **Map** har bla. metoder *has*, *set*, *get* og *delete*.
- **Set** har bla. metoder *has*, *add* og *delete*.

Oppgave 6 (40% ~ 96 minutter) – Web backend med Spring MVC

I denne oppgaven skal du jobbe litt med en liten web-applikasjon for å registrere interessen for ulike valgemner for høstsemesteret i 3. klasse.

Applikasjonen har kun én side (se bildet nedenfor) som brukes både til å registrere interesse, og til å se resultatet så langt.

Bruker registrerer interesse for emne(r) ved å huke av i én eller flere checkbox-er til venstre i tabellen og deretter trykke på knappen «Registrer dine valg!». Resultatet fra tidligere innsendte skjemaer vises til høyre i tabellen.



Valgemner høsten 2024

Vi ønsker at du krysser av nedenfor for de valgemner du er mest interessert i til høsten, slik at vi kan få en oversikt over interessen for de ulike emnene.

Valg	Kode	Navn	Stp	Antall
<input type="checkbox"/>	DAT150	Nettverksadministrasjon, drift og sikkerhet	10	5
<input type="checkbox"/>	DAT152	Videregående webapplikasjoner	10	8
<input type="checkbox"/>	DAT155	3D Datagrafikk	10	5
<input type="checkbox"/>	DAT156	Praksis i arbeidslivet for data	10	2
<input type="checkbox"/>	DAT158	Maskinlæring og videregående algoritmer	10	13
<input type="checkbox"/>	DAT160	Intelligente roboter	10	3
<input type="checkbox"/>	INF122	Funksjonell programmering	10	7

Du kan maks krysse av for 3 emner.

Registrer dine valg!

Validering, feilmelding og kvittering

Brukerinput skal valideres, dvs. at det er valgt 1-3 emner, og at alle valgte emner finnes i databasen og går i høstsemesteret.

Ved feil skal en generell feilmelding vises med rødt i toppen av siden, slik:



... (resten av siden er utelatt)

Etter gyldig innsending og registrering vises en kvittering med grønt i toppen av siden, slik:



... (resten av siden er utelatt)

Data i applikasjonen

Data om emner og interessen for disse er lagret i én enkelt tabell i en database, slik:

```
CREATE TABLE emne (  
  kode CHAR(6) PRIMARY KEY,      -- f.eks. 'DAT100'  
  navn VARCHAR,                  -- f.eks. 'Grunnleggende programmering'  
  studiepoeng INTEGER,           -- f.eks. 10  
  semester CHAR(1),              -- 'v'|'h'|'b' for vår, høst eller begge  
  type CHAR(1),                  -- 'o'|'v' for obligatorisk eller valgfag  
  antall INTEGER                 -- antall interesserte  
);
```

Det er definert en tilsvarende entitetsklasse i Java, slik:

```
@Entity
public class Emne {

    @Id private String kode;
    private String navn;
    private Integer studiepoeng;
    private String semester;
    private String type;
    private Integer antall;

    public void registrerInteresse() { antall++; }

    // + gettere for alle egenskaper
}
```

Lagdeling og komponenter i applikasjonen

Datatilgang gjøres via et

```
interface EmneRepository extends JpaRepository<Emne, String>
```

der de interessante metodene for oss er

```
List<Emne> findAll()
Emne findById(String emnekode)
```

Mellom controller og repository skal det legges en

```
@Service class EmneRepoService
```

med metodene

```
List<Emne> finnValgemnerForSemester(String semester) // "v" eller "h"
void registrerInteresseFor(List<String> emnekoder)
```

Controlleren

```
@Controller class EmnelisteController
```

skal håndtere følgende requests

```
@GetMapping("/valgemner") // Request om å se nettsiden. Samarbeider
                           // med viewet kalt "emneliste".

@PostMapping("/registrer") // Request om å registrere interesse for
                           // emner. Etterpå vises emnelisten igjen.
```

Arkitekturkrav ellers

- Løsningen skal utformes i Spring MVC
- Applikasjonen skal bruke Model-View-Controller (MVC)
- View-komponenter skal skrives med Java ServerPages (JSP)
- JSP skal bruke Expression Language (EL) og Standard Tag Library (JSTL)
- Det skal benyttes Post-Redirect-Get ved POST-requests

Oppgaver

a – 10%)

Skriv koden for `emneliste.jsp`.

- Det er tilstrekkelig å kun skrive `<body>`-taggen og innholdet i denne.
- Øverst på siden skal det kunne vises feilmelding/kvittering, om aktuelt.
- Det er ikke nødvendig å sette opp emnelisten i en `<table>`. Det gir full score å sette den opp med å bruke en `<p></p>` per linje.
- Tips: `<input type="checkbox" ... >` vil ha samme navn, men ulik verdi for de ulike valgene
- En del elementer i JSP-en har direkte sammenheng med hvordan ting gjøres i controlleren, og det er viktig at det er samsvar mellom JSP og controller (neste spørsmål).

b – 15%)

Skriv koden for `EmnelisteController`, inkl. metoder for GET- og POST-mappingene som er nevnt i oppgaveteksten over, samt en metode for validering av input.

- Pass på at løsningen samsvarer med JSP-en (forrige spørsmål).
- Ved innsending av skjema:
 - Request-parametre for checkboxen kommer inn som en List av verdier.
 - Ved ugyldig input skal det ikke registreres noe i databasen, men lages til en generell feilmelding som deretter vises på siden.
 - Ved gyldig input skal valgene registreres i databasen, og det skal lages en kvitteringsmelding som deretter vises på siden.
- Metoden for validering av input skal ha denne signaturen:
`boolean erGyldigEmnevalg(List<String> emnevalg)`
Gyldig liste av emnevalg er en liste av 1-3 emnekoder som alle er registrerte høstvalgfag i databasen. Tips for å sjekke match med databasedata: Du kan spørre en List om den `containsAll` elementer i en annen List.

c – 10%)

Skriv koden for `EmneRepoService`

- `List<Emne> finnValgemnerForSemester(String semester)` skal returnere alle valgemner (type 'v') for gitt semester. Emnelisten skal være sortert på emnekode. Du får pluss for å bruke streams i løsningen.
- `void registrerInteresseFor(List<String> emnekoder)` skal oppdatere databasen med registrert interesse for emnene for gitt liste av emnekoder.

d – 5%)

Vi ønsker å enhetsteste metoden for inputvalidering `erGyldigEmnevalg(List<String> emnevalg)`. Skriv en klasse `InputvalideringTest` for denne enhetstesting. Tips: Du kan bruke mockito-rammeverket for å erstatte `EmneRepoService`-en som gir deg databasedata å sammenligne med. Du kan anta at `Emne`-klassen har en parametrisk konstruktør for enkel opprettelse av emne-objekter.