

EKSAMENSOPPGAVE

Emnekode: DAT108

Emnenavn: Programmering og webapplikasjoner

Klasse: 2. klasse DATA / INF

Dato: 16. juni 2023

Eksamensform: Skriftlig digital skoleeksamen (Wiseflow | FLOWlock)

Eksamenstid: 4 timer (0900-1300)

Antall eksamensoppgaver: 4

Antall sider (medregnet denne): 10

Antall vedlegg: Ingen

Tillatte hjelpemiddel: Ingen

Lærere: Lars-Petter Helland (928 28 046)
 Bjarte Kileng (909 97 348)

LYKKE TIL!

Oppgave 1 (20% ~ 48 minutter) – Lambda-uttrykk og strømmer

For å få full score må løsningene ikke være unødige kompliserte, og det bør brukes metodereferanser der det er mulig.

- a) Nedenfor ser du 5 ulike λ -uttrykk, et par uttrykt som metodereferanse. Skriv én setning for hver av disse der de tilordnes en variabel. Det vi er ute etter er datatypen til variablene, f.eks. om det er en `Consumer<String>`, en `Function<Integer, String>`, etc. ..

```
h -> h >= 0 && h <= 59      // der h er et heltall
b -> !b                      // der b er en boolsk verdi
s -> s.startsWith("A")      // der s er en string
String::length
System.out::println         // Her er det flere muligheter. Velg én.
```

- b) Nå skal vi se på en liste av eksamensobjekter. En eksamen har **emnekode**, **dato** (`LocalDate`) og **antallOppmeldte**. Vi ønsker å sortere listen kronologisk (dvs. sortert etter eksamensdato), og deretter skrive den ut på skjermen. Det er 3 punkter i løsningen (nummerert i koden), som mangler, og som du må svare på:

1. Hva er **datatypen til komparatoren** kronologisk, som blir 2. parameter i `sort(...)`?
2. Skriv **kode for denne komparatoren** som vil gi kronologisk sortering.
3. Skriv **kode for utskriften, én eksamen per linje**. Anta at Eksamen har en `toString()`. Du får ikke bruke løkke!

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

    List<Eksamen> eksamener = Arrays.asList(
        new Eksamen("DAT102", LocalDate.of(2023, Month.MAY, 30), 150),
        new Eksamen("DAT107", LocalDate.of(2023, Month.MAY, 15), 160),
        new Eksamen("DAT108", LocalDate.of(2023, Month.JUNE, 6), 50);

    ### 1 ### kronologisk = ### 2 ###
    Collections.sort(eksamener, kronologisk);
    ### 3 ###
}

--

Resultat av kjøring:
Eksamen [emnekode=DAT107, dato=2023-05-15, antallOppmeldte=160]
Eksamen [emnekode=DAT102, dato=2023-05-30, antallOppmeldte=150]
Eksamen [emnekode=DAT108, dato=2023-06-06, antallOppmeldte=50]
```

- c) Nedenfor ser du en metode **fikse()**, en metode **main()** der denne blir kalt, og resultatet av å kjøre main(). Programmet "fikser" store og små bokstaver i en liste av strenger, slik at alle strengene blir på formen "Aaaaa".

```
List<String> fikse(List<String> liste, ### 1 ### f) {
    return liste.stream().map(f).toList();
}

void main(...) {
    List<String> liste = List.of("ola", "Per", "pÅL", "ESPEN");

    ### 1 ### navnefiks = ### 2 ###
    List<String> resultat = fikse(liste, navnefiks);
    System.out.println(resultat);
}

--

Resultat av kjøring:
[Ola, Per, Pål, Espen]
```

1. Hva skal stå der det står **### 1 ###** (dvs. typen til parameteren f, som er det samme som typen til variabelen navnefiks i main)?
2. Hva skal stå der det står **### 2 ###** (dvs. definisjonen av navnefiks)? *Tips: s.substring(0,1) gir første bokstav i en streng, og s.substring(1, s.length()) gir resten av strengen. Disse kan brukes i løsningen.)*

I oppgavene d)-f) skal vi jobbe litt mer med listen av eksamener fra b)

```
List<Eksamen> eksamener = Arrays.asList(
    new Eksamen("DAT102", LocalDate.of(2023, Month.MAY, 30), 150),
    new Eksamen("DAT107", LocalDate.of(2023, Month.MAY, 15), 160),
    new Eksamen("DAT108", LocalDate.of(2023, Month.JUNE, 6), 50),

    ... vi får anta at det er noen 100 rader til ...

);
```

- d) Bruk streams til å lage en ny liste av alle eksamenene i DAT108 som er i juni (Month.JUNE), dvs. alle DAT108-konteeksamener. Legg svaret inn i en variabel.
- e) Bruk streams til å skrive ut på skjermen (uten duplikater) alle **emnekode**er det er registrert eksamen i, én linje per emne.
- f) Bruk streams til å beregne gjennomsnittlig antall oppmeldte i alle DAT108-konteeksamener. Bruk helst svaret fra d) i løsningen din. Kutt evt. desimaler slik at svaret blir et heltall. Legg svaret inn i en variabel.

Oppgave 2 (25% ~ 60 minutter) – JavaScript

- a) JavaScript og «strict mode».
- Hva er konsekvensene av «strict mode» i JavaScript?
 - Hvordan kan vi instruere JavaScript-motoren til å benytte «strict mode»?
 - JavaScript-motoren vil kunne benytte «strict mode» uten at det er angitt i koden. Gi et eksempel.

- b) JavaScript *string template*, og litt om **Date** objektet:

- i. Koden nedenfor oppretter en JavaScript *string template*:

```
const info = `${navn} ble født på en ${dag}.`;
```

Ta utgangspunkt i koden over forklar bruken av en *string template*. Hva betyr konstruksjonene *\${navn}* og *\${dag}*?

- ii. En variabel *dag* er opprettet ved følgende kode:

```
const dag = new Date(Date.UTC(2003,0,12)).toLocaleDateString(navigator.language, {
  weekday: 'long'
});
```

Hva gjør koden over? Hva slags type data vil *dag* inneholde? Hva gjør *navigator.language* i koden over?

- c) En JavaScript-klasse **Karakterer** skal lages for å registrere karakterer i et fag i et semester. Konstruktøren tar to parametere, emnekode til faget, og semester for eksamen

Klassen **Karakterer** har bla. følgende offentlige (public) metoder:

- Metode *addStudent (student)*: Legger inn karakter for en student.

Parametre:

- Inn-parameter: **object**
- Returverdi: **object** eller verdien *null*

Inn-parameter *student* er et objekt med egenskaper *id*, *etternavn*, *fornavn* og *karakter*:

- Attributtet *id* er av type **string**.
 - En *id* må starte med et bokstavtegn, så 3 sifre, og er unik for hver student.
- Attributtene *etternavn* og *fornavn* er av type **string**.
 - Ellers ingen krav.
- Attributtet *karakter* er av type **string**.
 - Må bestå av ett enkelt tegn, A til F.

Dersom formatet på inn-parameter *student* er feil eller en student med gitt *id* allerede finnes blir ikke karakteren registrert.

Returverdi:

- Hvis karakteren blir registrert returneres inn-parameter *student*.
- Hvis karakteren ikke blir registrert returneres verdien *null*.

- Metode *getStatistikk()*: Returnerer karakterstatistikk for emnet.

Parametere:

- Inn-parameter: Ingen inn-parametere
- Returverdi: **object**

Returverdi skal være et objekt med følgende egenskaper:

- Attributt *emne* er emnekode til faget.
- Attributt *semester* er semester for eksamen.
- Attributt karakterfordeling er et objekt som viser karakterfordelingen på eksamen

Kodeeksempelet nedenfor demonstrerer bruk av **Karakterer**:

```
const karakterliste = new Karakterer("DAT108", "V2023");

karakterliste.addStudent({
  "id": "o123",
  "fornavn": "Ole",
  "etternavn": "Olsen",
  "karakter": "B"
});

karakterliste.addStudent({
  "id": "g324",
  "fornavn": "Gro",
  "etternavn": "Grosen",
  "karakter": "B"
});

const statistikk = karakterliste.getStatistikk();
```

Etter at koden har kjørt skal objektet *statistikk* ha følgende innhold:

```
{
  "emne": "DAT108",
  "semester": "V2023",
  "karakterfordeling": {
    "A": 0,
    "B": 2,
    "C": 0,
    "D": 0,
    "E": 0,
    "F": 0
  }
}
```

Oppgave: Skriv JavaScript-koden for **Karakterer** i samsvar med teksten over. (PS! Neste side har en del tips/hjelp.)

Hjelp:

- Kodeeksempelet nedenfor sjekker om en variabel *karakter* er en gyldig karakter:

```
if (/^[A-F]$/.test(karakter)) console.log("Gyldig karakter");
```
- Et regulært uttrykk for en gyldig student ID, i henhold til oppgaveteksten er gitt nedenfor:

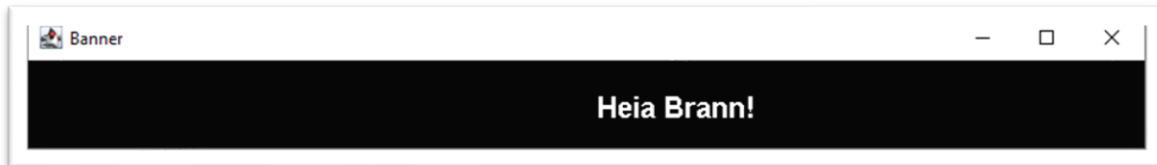
```
/^\p{Letter}\d{3}$/u
```
- Både **Map**, **Set** og **Array** har en metode *forEach*.
- **Map** og **Set** sin metode *values* returnerer en iterator over alle verdier.
- **Map** sin metode *keys* returnerer en iterator over alle nøkler.
- Iteratorer, **Array**, **Map**, **Set** og **String** kan gjennomløpes med en *for...of* setning:

```
const karakterfordeling = {};  
for (const k of "ABCDEF") {  
  karakterfordeling[k] = 0;  
}
```
- **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*.
- Operator *typeof* returnerer en **string** som angir operanden sin type.
- Operator *instanceof* tester om operand er en forekomst av en klasse, også via arv.

Oppgave 3 (15% ~ 36 minutter) – Tråder

Vi ønsker å kontrollere en rulletekst som skal vises på en skjerm. Vi har en ferdiglaget klasse kalt **Banner** som vi skal bruke i løsningen vår. Her er koden for å få et slikt Banner opp på skjermen.

```
Banner banner = new Banner("Heia Brann!");
```



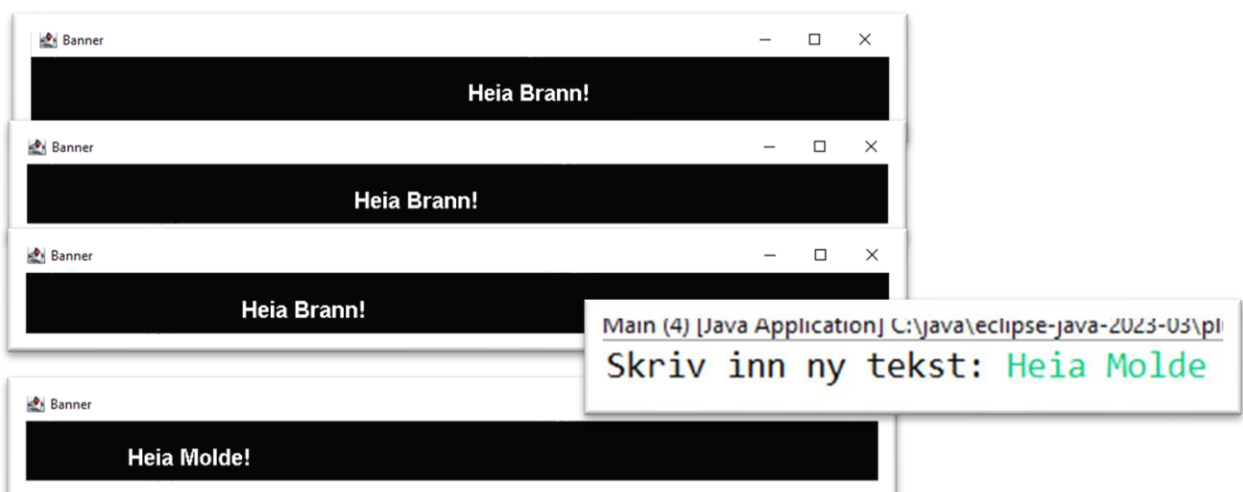
Men vi ønsker at det skal rulle, og vi ønsker å kunne endre teksten mens det ruller. Banner har et par ferdige metoder vi kan bruke i løsningen vår.

```
/** Setter ny tekst på banneret */  
public void setText(String text);  
  
/** Flytter teksten en pixel til venstre. Hvis teksten kommer til enden  
 * tegnes den automatisk opp til høyre ved neste flytt. */  
public void repaint();
```

Oppgaven din er å lage en trådklasse som inneholder et slikt banner, og som bruker det til å vise en rullende tekst. For å få en grei animasjon kan du f.eks. vente **10 ms** mellom hver **repaint**.

Trådklassen bør gi (main) mulighet til å sette/endre teksten "i fart" ved å tilby en metode for dette.

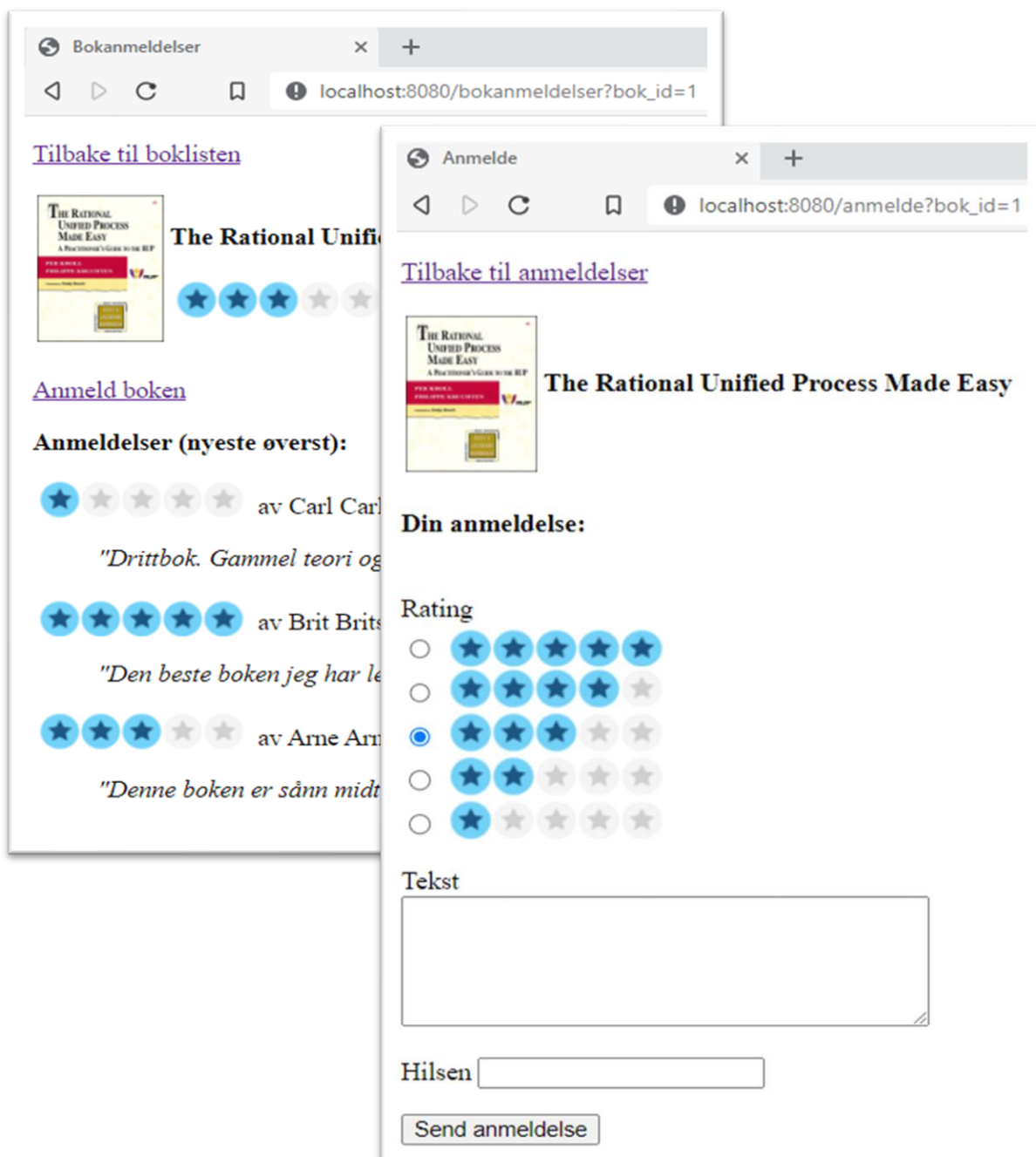
- Lag trådklassen som beskrevet over. Kall den **Rulletekst**.
- Lag et lite **main**-program som bruker denne trådklassen til å få opp en rulletekst med "Heia Brann!". Deretter har main en evig løkke som lar brukeren gi inn/endre tekst som "rulles".



Oppgave 4 (40% ~ 96 minutter) – Web backend med Spring MVC

I denne oppgaven skal du jobbe med og lage en liten del av en liten webapplikasjon for bokanmeldelser. Et par URL-er / sider i applikasjonen:

- **bokanmeldelser** (bakerste bilde) viser alle anmeldelser for en bok (med id = param **bok_id**).
- **anmelde** (fremste bilde) viser skjema for anmeldelse av en bok (med id = param **bok_id**). Ved trykk på knappen **[Send anmeldelse]** blir anmeldelsen postet til den samme URL-en. Dette fører til at anmeldelsen blir lagret, og bruker blir omdirigert til **bokanmeldelser** for boken.
- Ved feilsituasjoner skal det redirectes til en egen URL **feilmelding**. Eksempler på feil kan være at man gjør en request med manglende eller ugyldig **bok_id** (dvs. hvis man ikke klarer å gjøre et vellykket søk etter bok med gitt bok_id)



Oppbygging av applikasjonen

I bunn av applikasjonen har vi en database med to tabeller, bok og anmeldelse. Disse er også representert i Java med klassene **Bok** og **Anmeldelse**. De ser slik ut:

```
@Entity public class Bok {

    @Id private Integer id;
    private String tittel;
    private String bildefil; // Navnet på bildefilen

    @OneToMany(mappedBy = "bok", fetch = FetchType.EAGER)
    private List<Anmeldelse> anmeldelser;

    public void leggTilAnmeldelse(Anmeldelse anmeldelse)
    public int getAvrundetSnittrating() // gir avrundet snitt 1,2,3,4 eller 5

    ... diverse andre metoder ...

}

@Entity public class Anmeldelse {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private Integer rating; // 1 .. 5
    private String tekst;
    private String anmelder;

    @ManyToOne @JoinColumn(name = "bok_id")
    private Bok bok;

    ... diverse metoder ...

}
```

På toppen av dette har vi definert to repositories som Spring da vil generere nødvendig kode for:


```
public interface BokRepo extends JpaRepository<Bok, Integer> {}

public interface AnmeldelseRepo extends JpaRepository<Anmeldelse, Integer> {}
```

Du skal lage **én controller** og **én JSP** for URL-en **/anmelde** med den funksjonaliteten det spørres etter i oppgavene, se nedenfor.

Du kan i teorien kalle metoder i **BokRepo** og **AnmeldelseRepo** direkte fra controlleren, men for å få full score må du også lage en service mellom controller og repositories slik at controlleren går via denne i stedet for rett mot repo-ene.

For å få full score må du også løse oppgavene på best mulig måte i hht. prinsippene i kurset, dvs. **Model-View-Controller**, **Post-Redirect-Get**, **Expression Language (EL)** og **JSTL** i JSP-ene, elegant kode, osv ...

- a) Vi starter med **servicen** mellom controller og repositories. Vi kan kalle denne **BokAnmeldelseService**. Funksjonalitet den skal ha er i) **å finne en bok med en gitt id**, og ii) **å lagre en bokanmeldelse**. Skriv denne.
- b) Skriv metoden i controlleren som behandler en GET-requester til **/anmelde**. Som du ser av URL-en på skjermbildet (http://localhost:8080/anmelde?bok_id=1) er det en request-parameter **bok_id** som angir hvilken bok det gjelder. Ved manglende eller ugyldig bok_id skal det omdirigeres til feilmelding. Ellers skal siden vises som i skjermbildet.
- c) Skriv jsp-en for **anmelde-siden**. Du trenger ikke bry deg om JSP-direktiver. Du kan anta at bilder ligger lagret i mappen **bilder** i applikasjonen. Navn på bildefil ligger som en egenskap i Bok. Bildene som angir stjerner for rating heter ratingX.png, f.eks. **** for bildet  (3 stjerner). For full score skal det brukes en løkke for å sette opp de 5 rating-alternativene. (Tips: JSTL har en for-løkke som ser slik ut, `<c:forEach var="i" begin="1" end="100">`.) Rating  (3 stjerner) skal være checked i utgangspunktet.
- d) Skriv metoden i controlleren som behandler requesten til **/anmelde** som er resultatet av at brukeren trykker på knappen **[Send anmeldelse]**. Du kan anta at forespørselen kommer fra skjemaet, dvs. du trenger ikke å gjøre håndteringen robust ut over det. Hvis anmelderfeltet er tomt, skal anmeldelsen lagres med anmelder "Anonym". Etter anmeldelsen er lagret skal man omdirigeres til **bokanmeldelser** for aktuell bok.
- e) Vi ønsker å ha mulighet til forhåndsutfylling av anmelder-feltet hvis noen har anmeldt en bok fra samme nettleser tidligere. Til dette kan man bruke informasjonskapsler (cookies). Vis med frittstående kodesnutter (dvs. ikke bland sammen med tidligere svar) hvilken kode som må legges til/endres i b), c) og d) for å få denne tilleggfunksjonaliteten.

EKSAMENSOPPGÅVE

Emnekode: DAT108

Emnenamn: Programmering og webapplikasjoner

Klasse: 2. klasse DATA / INF

Dato: 16. juni 2023

Eksamensform: Skriftleg skuleeksamen (Wiseflow | FLOWlock)

Eksamenstid: 4 timer (0900-1300)

Tal på eksamensoppgåver: 4

Tal på sider (medrekna denne): 10

Tal på vedlegg: Ingen

Tillatte hjelpemiddel: Ingen

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

LYKKE TIL!

Oppgave 1 (20% ~ 48 minutt) – Lambda-uttrykk og straumer

For å få full score må løysingane ikkje vera unødige kompliserte, og det bør brukast metodereferansar der det er mogleg.

- a) Nedanfor ser du 5 ulike λ -uttrykk, eit par uttrykt som metodereferanse. Skriv éi setning for kvar av desse der dei blir tilordna ein variabel. Det vi er ute etter er datatypen til variablane, t.d. om det er ein `Consumer<String>`, ein `Function<Integer, String>`, etc. ..

```
h -> h >= 0 && h <= 59      // der h er et heltall
b -> !b                      // der b er en boolsk verdi
s -> s.startsWith("A")      // der s er en string
String::length
System.out::println         // Her er det flere muligheter. Velg én.
```

- b) No skal me sjå på ei liste av eksamensobjekt. Ein eksamen har **emnekode**, **dato** (`LocalDate`) og **antallOppmeldte**. Vi ønskjer å sortere lista kronologisk (dvs. sortert etter eksamensdato), og deretter skrive ho ut på skjermen. Det er 3 punkt i løysninga (nummerert i koden), som manglar, og som du må svare på:
1. Kva er **datatypen til komparatoren** kronologisk, som blir 2. parameter i `sort(...)`?
 2. Skriv **kode for denne komparatoren** som vil gi kronologisk sortering.
 3. Skriv **kode for utskrifta, ein eksamen per linje**. Gå ut frå at Eksamen har ein `toString()`. Du får ikkje bruke `løkke`!

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

    List<Eksamen> eksamener = Arrays.asList(
        new Eksamen("DAT102", LocalDate.of(2023, Month.MAY, 30), 150),
        new Eksamen("DAT107", LocalDate.of(2023, Month.MAY, 15), 160),
        new Eksamen("DAT108", LocalDate.of(2023, Month.JUNE, 6), 50);

    ### 1 ### kronologisk = ### 2 ###
    Collections.sort(eksamener, kronologisk);
    ### 3 ###
}

--

Resultat av kjøring:
Eksamen [emnekode=DAT107, dato=2023-05-15, antallOppmeldte=160]
Eksamen [emnekode=DAT102, dato=2023-05-30, antallOppmeldte=150]
Eksamen [emnekode=DAT108, dato=2023-06-06, antallOppmeldte=50]
```

- c) Nedanfor ser du en metode **fikse()**, ein metode **main()** der denne blir kalla, og resultatet av å køyre main(). Programmet "fiksar" store og små bokstaver i ei liste av strenger, slik at alle strengene blir på forma "Aaaaa".

```
List<String> fikse(List<String> liste, ### 1 ### f) {
    return liste.stream().map(f).toList();
}

void main(...) {
    List<String> liste = List.of("ola", "Per", "pÅL", "ESPEN");

    ### 1 ### navnefiks = ### 2 ###
    List<String> resultat = fikse(liste, navnefiks);
    System.out.println(resultat);
}

--

Resultat av kjøring:
[Ola, Per, Pål, Espen]
```

1. Kva skal stå der det står **### 1 ###** (dvs. typen til parameteren f, som er det same som typen til variabelen navnefiks i main)?
2. Kva skal stå der det står **### 2 ###** (dvs. definisjonen av navnefiks)? *Tips: s.substring(0,1) gir første bokstav i en streng, og s.substring(1, s.length()) gir resten av strengen. Desse kan brukast i løysinga.)*

I oppgåvene d)-f) skal vi jobbe litt meir med lista av eksamenar frå b)

```
List<Eksamen> eksamener = Arrays.asList(
    new Eksamen("DAT102", LocalDate.of(2023, Month.MAY, 30), 150),
    new Eksamen("DAT107", LocalDate.of(2023, Month.MAY, 15), 160),
    new Eksamen("DAT108", LocalDate.of(2023, Month.JUNE, 6), 50),

    ... vi får anta at det er noen 100 rader til ...

);
```

- d) Bruk streams til å lage ei ny liste av alle eksamenane i DAT108 som er i juni (Month.JUNE), dvs. alle DAT108-konteeksamener. Legg svaret inn i ein variabel.
- e) Bruk streams til å skrive ut på skjermen (utan duplikat) alle **emnekode**er det er registrert eksamen i, éi linje per emne.
- f) Bruk streams til å berekne gjennomsnittleg tal på oppmeldte i alle DAT108-konteeksamener. Bruk helst svaret frå d) i løysninga di. Kutt evt. desimalar slik at svaret blir eit heiltal. Legg svaret inn i ein variabel.

Oppgave 2 (25% ~ 60 minutt) – JavaScript

- a) JavaScript og «strict mode».
- Kva er konsekvensane av «strict mode» i JavaScript?
 - Korleis kan vi instruere JavaScript-motoren til å nytte «strict mode»?
 - JavaScript-motoren vil kunne nytte «strict mode» utan at det er angitt i koden. Gje eit døme.

- b) JavaScript *string template*, og litt om **Date** objektet:

- i. Koden nedanfor skipar ein JavaScript *string template*:

```
const info = `${navn} ble født på en ${dag}.`;
```

Ta utgangspunkt i koden over forklår bruken av ein *string template*. Kva tyder konstruksjonane *\${navn}* og *\${dag}*?

- ii. Ein variabel *dag* er skipa ved følgjande kode:

```
const dag = new Date(Date.UTC(2003,0,12)).toLocaleDateString(navigator.language, {
  weekday: 'long'
});
```

Kva gjer koden over? Kva slags type data vil *dag* innehalde? Kva gjer *navigator.language* i koden over?

- c) Ei JavaScript-klasse **Karakterer** skal verte laga for å registrere karakterar i eit fag i eit semester. Konstruktøren tar to parameter, emnekode til faget, og semester for eksamen

Klassen **Karakterer** har bla. følgjande offentlege (public) metodar:

- Metode *addStudent (student)*: Legger inn karakter for ein student.

Parameter:

- Inn-parameter: **object**
- Returverde: **object** eller verde *null*

Inn-parameter *student* er eit objekt med eigenskapar *id*, *etternavn*, *fornavn* og *karakter*:

- Attributtet *id* er av type **string**.
 - Ein *id* må starta med eit bokstav teikn, så 3 siffer, og er unik for kvar student.
- Attributta *etternavn* og *fornavn* er av type **string**.
 - Inga fleire krav.
- Attributtet *karakter* er av type **string**.
 - Må bestå av eit enkelt teikn, A til F.

Dersom formatet på inn-parameter *student* er feil eller en student med gitt *id* allereie finnes blir ikkje karakteren registrert.

Returverde:

- Om karakteren verte registrert verte inn-parameter *student* returnert.
- Om karakteren ikkje verte registrert verte verde *null* returnert.

- Metode *getStatistikk()*: Returnerer karakterstatistikk for emnet.

Parameter:

- Inn-parameter: Inga inn-parameter
- Returverde: **object**

Returverde skal være eit objekt med følgjande eigenskapar:

- Attributt *emne* er emnekode til faget.
- Attributt *semester* er semester for eksamen.
- Attributt karakterfordeling er eit objekt som syner karakterfordelinga på eksamen

Døme nedanfor demonstrerer bruk av **Karakterer**:

```
const karakterliste = new Karakterer("DAT108", "V2023");

karakterliste.addStudent({
  "id": "o123",
  "fornavn": "Ole",
  "etternavn": "Olsen",
  "karakter": "B"
});

karakterliste.addStudent({
  "id": "g324",
  "fornavn": "Gro",
  "etternavn": "Grosen",
  "karakter": "B"
});

const statistikk = karakterliste.getStatistikk();
```

Etter at koden har køyrd skal objektet *statistikk* ha følgjande innhald:

```
{
  "emne": "DAT108",
  "semester": "V2023",
  "karakterfordeling": {
    "A": 0,
    "B": 2,
    "C": 0,
    "D": 0,
    "E": 0,
    "F": 0
  }
}
```

Oppgåve: Skriv JavaScript-koden for **Karakterer** i samsvar med teksten over. (PS! Neste side har ein del tips/hjelp.)

Hjelp:

- Døme med kode nedanfor sjekkar om ein variabel *karakter* er ein gyldig karakter:

```
if (/^[A-F]$/.test(karakter)) console.log("Gyldig karakter");
```
- Eit regulært uttrykk for ein gyldig student ID, i samsvar med oppgåveteksten er gitt nedanfor:

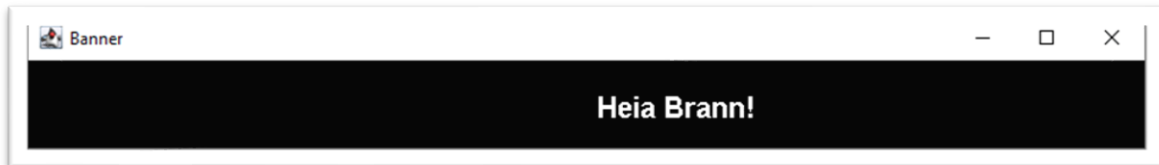
```
/^\p{Letter}\d{3}$/u
```
- Både **Map**, **Set** og **Array** har ein metode *forEach*.
- **Map** og **Set** sin metode *values* returnerer ein iterator over alle verdi.
- **Map** sin metode *keys* returnerer ein iterator over alle nøklar.
- Iteratorer, **Array**, **Map**, **Set** og **String** kan verte laupt gjennom med ein *for...of* setning:

```
const karakterfordeling = {};  
for (const k of "ABCDEF") {  
  karakterfordeling[k] = 0;  
}
```
- **Array** sin statiske metode *from*, og også spreie (*spread*) operator kan kopiere eit itererbart objekt til ein **Array**.
- **Map** har bla. metodar *has*, *set*, *get* og *delete*.
- **Set** har bla. metodar *has*, *add* og *delete*.
- Operator *typeof* returnerer ein **string** som angir operanden sin type.
- Operator *instanceof* testar om operand er ein førekomst av ei klasse, også via arv.

Oppgave 3 (15% ~ 36 minutt) – Trådar

Vi ønskjer å kontrollere ein rulletekst som skal visast på ein skjerm. Vi har ei ferdiglaga klasse kalla **Banner** som vi skal bruke i løysinga vår. Her er koden for å få eit slikt Banner opp på skjermen.

```
Banner banner = new Banner("Heia Brann!");
```



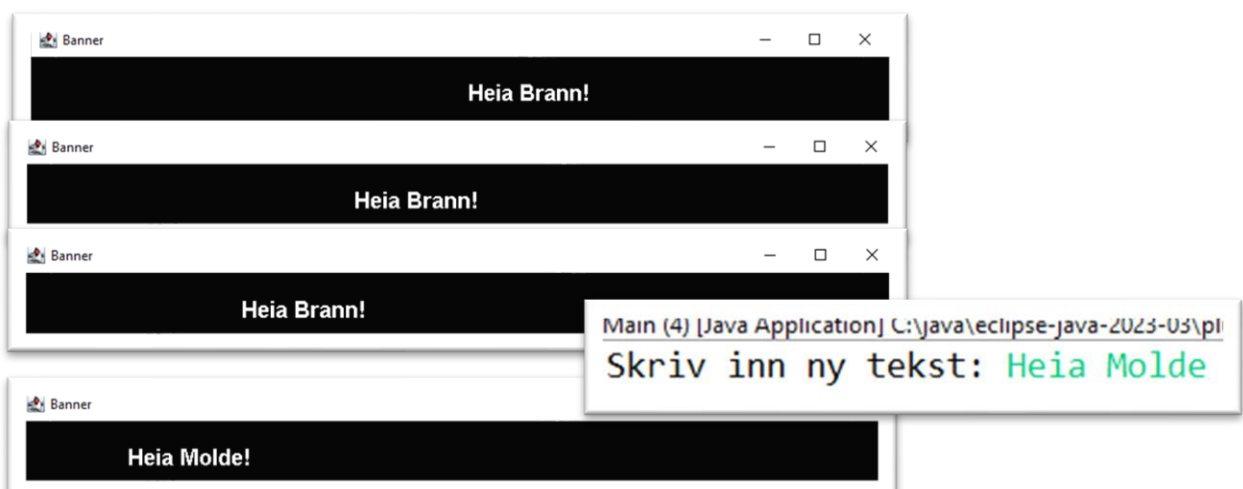
Men vi ønskjer at det skal rulle, og vi ønskjer å kunne endre teksten medan det rullar. Banner har eit par ferdige metodar vi kan bruke i løysinga vår.

```
/** Setter ny tekst på banneret */  
public void setText(String text);  
  
/** Flytter teksten en pixel til venstre. Hvis teksten kommer til enden  
 * tegnes den automatisk opp til høyre ved neste flytt. */  
public void repaint();
```

Oppgåva di er å lage ein trådklasse som inneheld eit slikt banner, og som bruker det til å vise ein rullande tekst. For å få ein grei animasjon kan du t.d. vente **10 ms** mellom kvar **repaint**.

Trådklassen bør gi (main) høve til å setje/endre teksten "i fart" ved å tilby ein metode for dette.

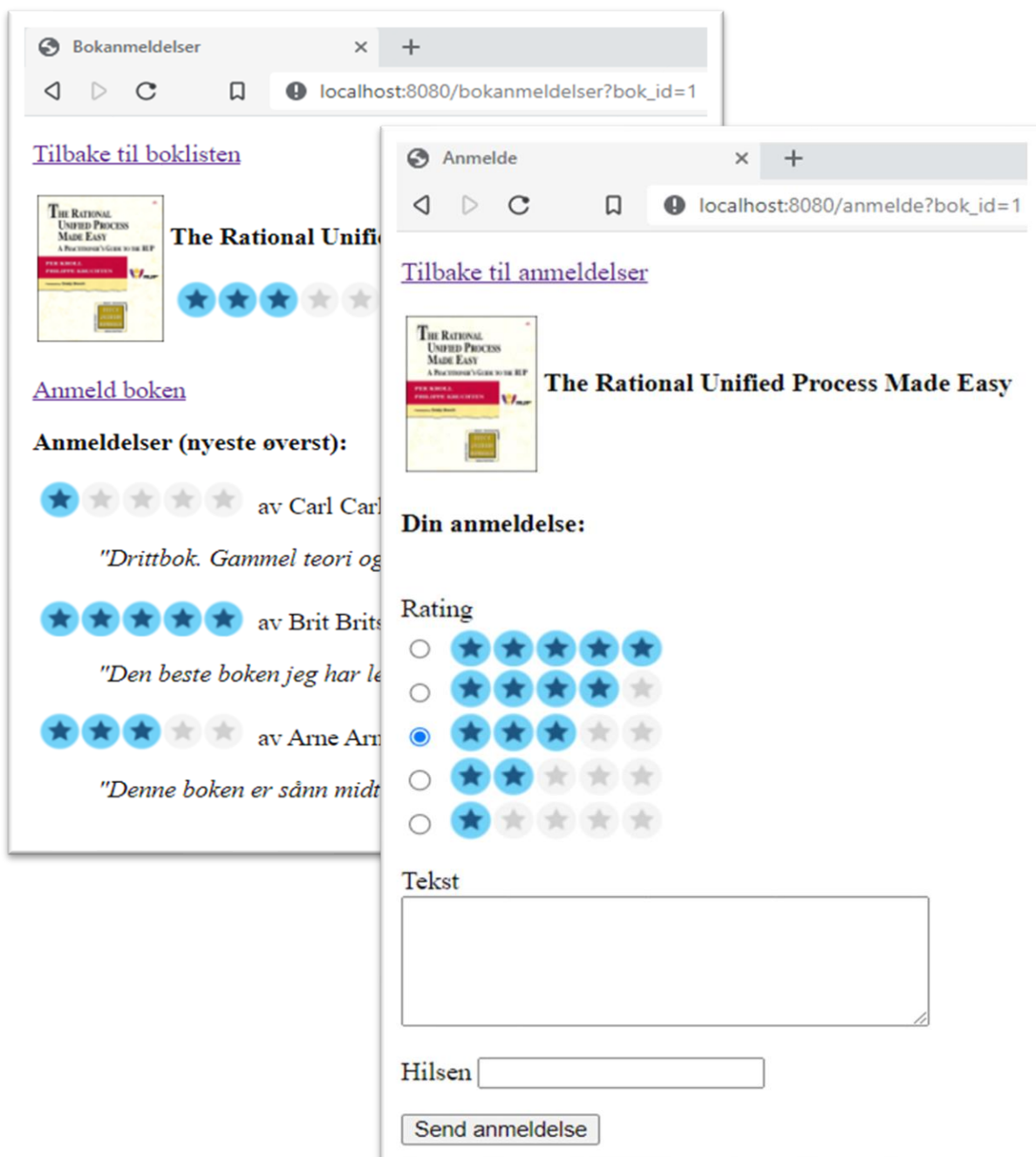
- Lag trådklassen som beskrive over. Kall den **Rulletekst**.
- Lag eit lite **main**-program som bruker denne trådklassen til å få opp ein rulletekst med "Heia Brann!". Deretter har main ei evig løkke som lèt brukaren gi inn/endre tekst som "blir rulla".



Oppgave 4 (40% ~ 96 minutt) – Web backend med Spring MVC

I denne oppgava skal du jobbe med og lage ein liten del av en liten webapplikasjon for bokmeldingar (bokmål: *bokanmeldelser*). Eit par URL-er / sider i applikasjonen:

- **bokanmeldelser** (bakarste bilete) viser alle meldingar for ei bok (med id = param **bok_id**).
- **anmelde** (fremste bilete) viser skjema for melding av ei bok (med id = param **bok_id**). Ved trykk på knappen **[Send anmeldelse]** blir meldinga posta til den same URL-en. Dette fører til at meldinga blir lagra, og bruker blir omdirigert til **bokanmeldelser** for boken.
- Ved feilsituasjonar skal det redirectast til ein egen URL **feilmelding**. Døme på feil kan vere at ein gjer ein request med manglande eller ugyldig **bok_id** (dvs. viss ein ikkje klarer å gjere eit vellykka søk etter bok med gitt bok_id)



Oppbygging av applikasjonen

I botn av applikasjonen har vi ein database med to tabellar, bok og anmeldelse. Desse er også representerte i Java med klassane **Bok** og **Anmeldelse**. Dei ser slik ut:

```
@Entity public class Bok {

    @Id private Integer id;
    private String tittel;
    private String bildefil; // Navnet på bildefilen

    @OneToMany(mappedBy = "bok", fetch = FetchType.EAGER)
    private List<Anmeldelse> anmeldelser;

    public void leggTilAnmeldelse(Anmeldelse anmeldelse)
    public int getAvrundetSnittrating() // gir avrundet snitt 1,2,3,4 eller 5

    ... diverse andre metoder ...

}

@Entity public class Anmeldelse {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private Integer rating; // 1 .. 5
    private String tekst;
    private String anmelder;

    @ManyToOne @JoinColumn(name = "bok_id")
    private Bok bok;

    ... diverse metoder ...

}
```

På toppen av dette har vi definert to repositories som Spring då vil generere nødvendig kode for:

```
public interface BokRepo extends JpaRepository<Bok, Integer> {}

public interface AnmeldelseRepo extends JpaRepository<Anmeldelse, Integer> {}
```

Du skal lage **én controller** og **én JSP** for URL-en **/anmelde** med den funksjonaliteten det blir spurt etter i oppgåvene, sjå nedanfor.

Du kan i teorien kalle metodar i **BokRepo** og **AnmeldelseRepo** direkte frå controlleren, men for å få full score må du også lage ein service mellom controller og repositories slik at controlleren går via denne i staden for rett mot repo-ene.

For å få full score må du også løyse oppgåvene på best mogeleg måte i hht. prinsippa i kurset, dvs. **Model-View-Controller**, **Post-Redirect-Get**, **Expression Language (EL)** og **JSTL** i JSP-ene, elegant kode, osb ...

- a) Vi starter med **servicen** mellom controller og repositories. Vi kan kalle denne **BokAnmeldelseService**. Funksjonalitet han skal ha er i) **å finne ei bok med ein gitt id**, og ii) **å lagre ei bokmelding** (*bokmål: bokanmeldelse*). Skriv denne.
- b) Skriv metoden i controlleren som behandler ein GET-requester til **/anmelde**. Som du ser av URL-en på skjermbiletet (http://localhost:8080/anmelde?bok_id=1) er det ein request-parameter **bok_id** som angir kva bok det gjelder. Ved manglande eller ugyldig bok_id skal det omdirigerast til feilmelding. Elles skal sida vises som i skjermbiletet.
- c) Skriv jsp-en for **anmelde-sida**. Du treng ikkje bry deg om JSP-direktiv. Du kan rekne med at bilete ligger lagra i mappa **bilder** i applikasjonen. Namn på biletil ligg som ein eigenskap i Bok. Bileta som angir stjerner for rating heiter ratingX.png, t.d. **** for biletet  (3 stjerner). For full score skal det brukast ei løkke for å setje opp dei 5 rating-alternativa. (Tips: *JSTL har ei for-løkke som ser slik ut, <c:forEach var="i" begin="1" end="100">*.) Rating  (3 stjerner) skal være checked i utgangspunktet.
- d) Skriv metoden i controlleren som behandlar requesten til **/anmelde** som er resultatet av at brukaren trykkjer på knappen **[Send anmeldelse]**. Du kan rekne med at førespurnaden kjem frå skjemaet, dvs. du treng ikkje å gjere handteringa robust ut over det. Viss anmelderfeltet er tomt, skal meldinga lagrast med anmelder "Anonym". Etter meldinga er lagra skal ein omdirigerast til **bokanmeldelser** for aktuell bok.
- e) Vi ønskjer å ha høve til førehandsutfylling av anmelder-feltet viss nokon har meldt ei bok frå same nettlesar tidlegare. Til dette kan ein bruke informasjonskapslar (cookies). Vis med frittstående kodesnuttar (dvs. ikkje bland saman med tidlegare svar) kva kode som må leggjast til/bli endra i b), c) og d) for å få denne tilleggsfunksjonaliteten.