

EKSAMENSOPPGAVE m/løsningsforslag

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 strømmmer

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.
```

Løsningsforslag:

```
2  Predicate<Integer> pi = h -> h >= 0 && h <= 59;
   //evt. Function<Integer, Boolean>, men kanskje litt unaturlig

2  //Litt ulike (likeverdige) alternativer ...
   Function<Boolean, Boolean> fbb = b -> !b;
   UnaryOperator<Boolean> uob = b -> !b;
   Predicate<Boolean> pb = b -> !b;

2  Predicate<String> ps = s -> s.startsWith("A");
   //evt. Function<String, Boolean>, men kanskje litt unaturlig

2  Function<String, Integer> fsi = String::length;

2  Consumer<String> cs = System.out::println;
   //..., men det trenger ikke være String.
```

- 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]
```

Løsningsforslag:

```
3    ### 1 ###: Comparator<Eksamen>  
3    ### 2 ###: (e1, e2) -> e1.getDatum().compareTo(e2.getDatum());  
    //evt. Comparator.comparing(Eksamen::getDatum)  
4    ### 3 ###: eksamener.forEach(System.out::println);  
    //Merk: Collections.sort(eksamener, kronologisk) sorterer eksamener  
    // slik at listen kan skrives rett ut etterpå.  
    //Hvis vi ser bort fra Collections.sort(eksamener, kronologisk) kan vi  
    // alternativt skrive ut sortert slik (sortere med sorted()):  
    // eksamener.stream().sorted(kronologisk).forEach(System.out::println);
```

- 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 ### navnefixs = ### 2 ###  
    List<String> resultat = fikse(liste, navnefixs);  
    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 navnefixs i main)?
2. Hva skal stå der det står **### 2 ###** (dvs. definisjonen av navnefixs)? *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.*

Løsningsforslag:

```
5    ### 1 ###: Function<String, String>  
5    ### 2 ###: s -> s.substring(0,1).toUpperCase()  
                + s.substring(1, s.length()).toLowerCase();
```

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-konteksamener. Legg svaret inn i en variabel.

Løsningsforslag:

```
2 List<Eksamen> dat108konts = eksamener.stream()
3+3 .filter(e -> e.getEmnekode().equals("DAT108")
    && e.getDato().getMonth() == Month.JUNE) //== OK pga Enum
2 .toList();

//evt.
3 Predicate<Eksamen> dat108iJuni = e -> e.getEmnekode().equals("DAT108")
    && e.getDato().getMonth() == Month.JUNE;
2+3+2 List<Eksamen> dat108konts = eksamener.stream().filter(dat108iJuni).toList();
```

- e) Bruk streams til å skrive ut på skjermen (uten duplikater) alle **emnekode**er det er registrert eksamen i, én linje per emne.

Løsningsforslag:

```
eksamener.stream()
3+3 .map(Eksamen::getEmnekode) //evt. e -> e.getEmnekode()
2 .distinct()
2 .forEach(System.out::println);
```

- 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.

Løsningsforslag:

Her er det flere alternativer som kan fungere. Det mest generelle er å bruke map-reduce, men vi kan også utnytte at IntStream har metoder både for sum og gjennomsnitt.

```
//Med map-reduce. Bruker ?: (if-else) for å unngå deling på 0.
```

```
int antall = dat108konter.size();
int sum = dat108konter.stream()
    .map(e -> e.getAntallOppmeldte())
    .reduce(0, (sum, e) -> sum + e);
int snitt = (antall == 0) ? 0 : (int) (sum / antall);
```

```
//Med mapToInt-sum. Bruker ?: (if-else) for å unngå deling på 0.
```

```
int antall = dat108konter.size();
int sum = dat108konter.stream()
    .mapToInt(e -> e.getAntallOppmeldte())
    .sum();
int snitt = (antall == 0) ? 0 : (int) (sum / antall);
```

```
//Med mapToInt-average. Må bruke Optional.orElse() pga. mulig deling med 0.
```

```
2 int snitt = (int) dat108konter.stream()
2+2     .mapToInt(e -> e.getAntallOppmeldte())
2       .average()
2       .orElse(0.0);
```

Oppgave 2 (25% ~ 60 minutter) – JavaScript

a) JavaScript og «strict mode».

i. Hva er konsekvensene av «strict mode» i JavaScript?

Løsningsforslag:

Denne modusen gjør at nettleser blir mer nøye med hvordan JavaScript-koden blir utført:

- Uten «strict mode» blir flere feil ignorert uten noen feil-melding. I «strict mode» vil koden feile.
- Med «strict mode» må alle variabler være deklartert.
- Diverse ord er reservert for JavaScript, og kan ikke brukes som variabelnavn, f.eks. **public**, **eval** etc.
- Når en metode kjøres utenfor konteksten av et objekt er størrelsen **this** ikke definert.
- Konstruksjoner med **with** er ulovlig.

ii. Hvordan kan vi instruere JavaScript-motoren til å benytte «strict mode»?

Løsningsforslag:

For å bruke «strict mode» for all koden, putt i starten av JavaScript koden: "use strict";

Koden kan også puttes i starten av en funksjon for å bruke «strict mode» kun for funksjonskoden.

iii. JavaScript-motoren vil kunne benytte «strict mode» uten at det er angitt i koden. Gi et eksempel.

Løsningsforslag:

For nye JavaScript-elementer vil ofte «strict mode» alltid være aktivt, og kan ikke slås av. Dette gjelder f.eks. for **WebWorkers**, inne i klasser, og ved import/export av moduler.

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}*?

Løsningsforslag:

Konstruksjonen *\${...}* er en plassholder, og vil erstattes av verdien inne i konstruksjonen. I dette eksempelet inneholder plassholderne variabler, og variablene sin verdi vil da bli satt inn i tekst-strengen. En plassholder kan også inneholde JavaScript-kode. Resultatet av å kjøre koden vil da bli satt inn i tekst-strengen.

Observer at variablene må ha verdi når *info* opprettes. Om variablene senere endrer verdi vil dette ikke endre *info*. Tekst-strengen til *info* er satt ut ifra verdiene som variablene hadde da *info* ble opprettet.

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?

Løsningsforslag:

Resultatene er:

- Variabelen *dag* vil være en **string**.
- Størrelsen ***navigator.language*** viser hvilket språk som brukes i nettleser, og som vanligvis er likt med språket som brukes på datamaskinen. For norsk bokmål vil denne strengen ha verdien ***nb-NO***.
- Koden vil finne ukedagen for 12. januar 2003, oversatt til språket som brukes i nettleser. Resultatet blir tilordnet til variabelen ***dag***. (Når koden har kjørt vil variabelen ***dag*** ha verdi ***søndag***, men det var det selvfølgelig ikke nødvendig å vite).

- c) En JavaScript-klasse **Karakterer** skal lages for å registrere karakterer i et fag i et semester. Konstruktøren tar to parametere, emnekoden 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 emnekoden til faget.
- Attributt *semester* er semester for eksamen.
- Attributt *karakterfordeling* er et objekt som viser karakterfordelingen på eksamen

Kodeeksemplet 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

Løsningsforslag:

```
class Karakterer {
  #emnekode;
  #semester;
  #studenter = new Map();

  constructor(emnekode, semester) {
    this.#emnekode = emnekode;
    this.#semester = semester;
  }

  addStudent(student) {
    const { id, fornavn, etternavn, karakter } = student;

    if (id === undefined) return null;
    const idreg = /^\\p{Letter}\\d{3}$/u;
    if (!idreg.test(id)) return null;

    if (fornavn === undefined) return null;
    if (etternavn === undefined) return null;
    if (karakter === undefined) return null;

    const karakterreg = /^[A-F]$/;
    if (!karakterreg.test(karakter)) return null;

    if (this.#hasStudent(id)) return null;

    this.#studenter.set(id, {
      "fornavn": fornavn,
      "etternavn": etternavn,
      "karakter": karakter
    });
    return student;
  }

  getStatistikk() {
    const karakterfordeling = {};
    for (const k of "ABCDEF") {
      karakterfordeling[k] = 0;
    }
    for (const student of this.#studenter.values()) {
      const karakter = student["karakter"];
      ++karakterfordeling[karakter];
    }

    const statistikk = {
      "emne": this.#emnekode,
      "semester": this.#semester,
      "karakterfordeling": karakterfordeling
    };
    return statistikk;
  }
}
```

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:

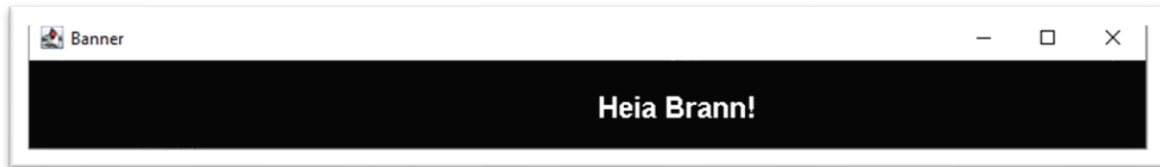
```
/^\p{Letter}\d{3}$/u
```
- Både **Map**, **Set** og **Array** har en metode *forEach*.
- **Map** og **Set** sin metode *values* returnerer en iterator over alle verdier.
- **Map** sin metode *keys* returnerer en iterator over alle nøkler.
- Iteratorer, **Array**, **Map**, **Set** og **String** kan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

a) Lag trådklassen som beskrevet over. Kall den **Rulletekst**.

Løsningsforslag:

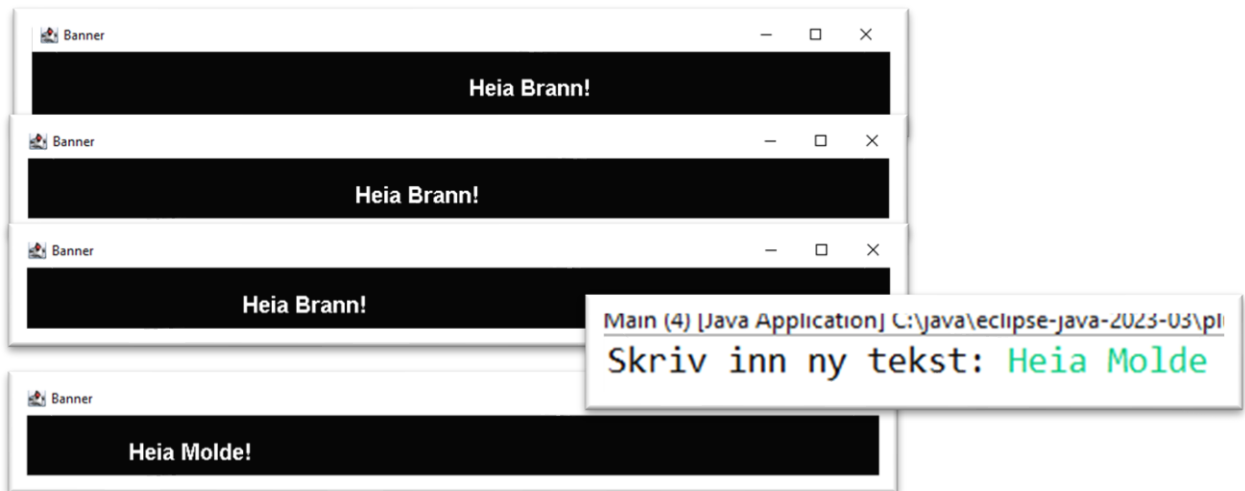
```
2 public class Rulletekst extends Thread { // implements Runnable også OK
2     private Banner banner;

    public Rulletekst(String tekst) { // Ikke nødv., men brukt i b)
        banner = new Banner(tekst);
    }

2     public void setTekst(String tekst) {
        banner.setText(tekst);
    }

    @Override
2     public void run() {
2         while (true) {
            banner.repaint();
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
            }
        }
    }
}
```

- b) 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".



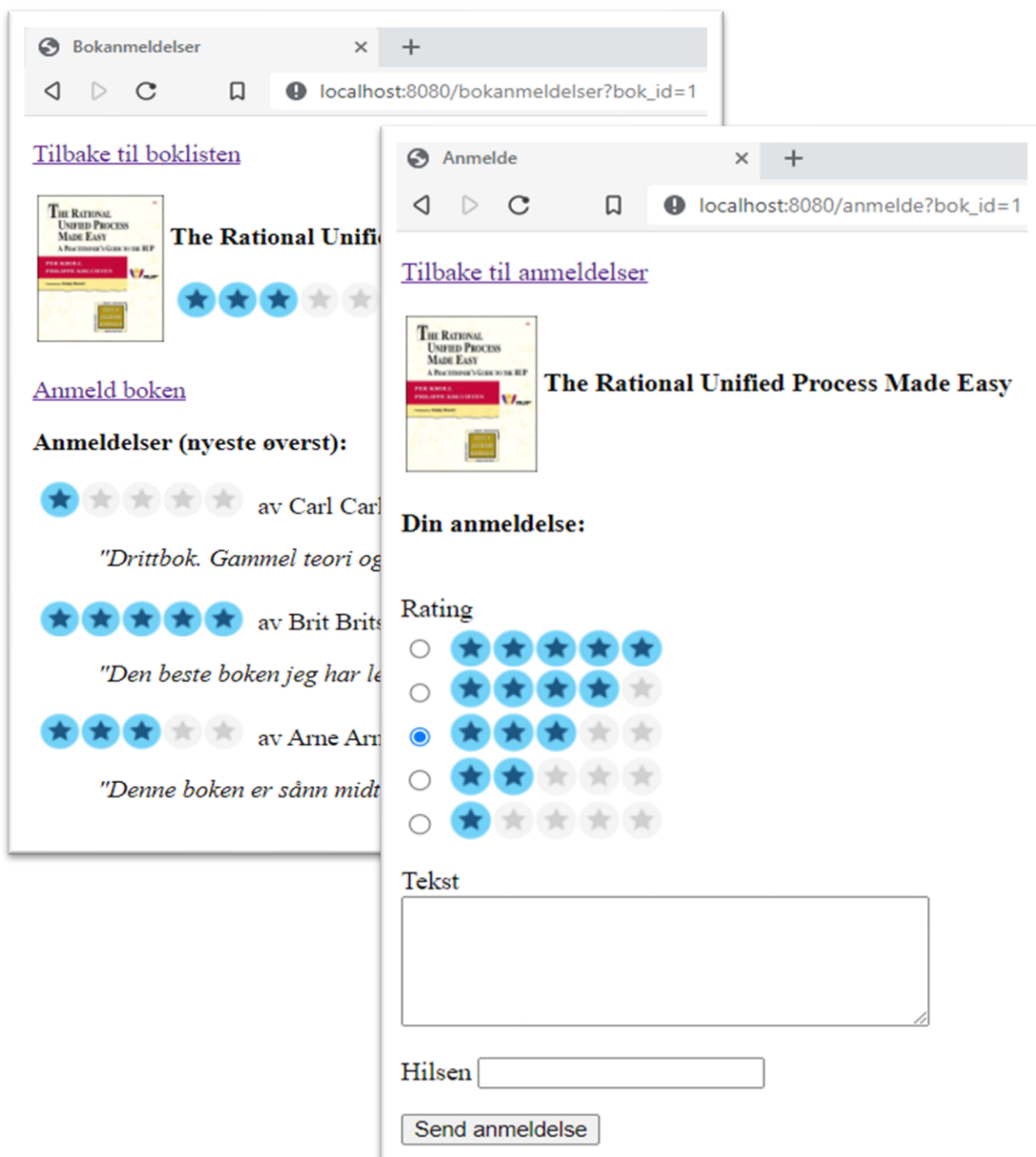
Løsningsforslag:

```
public static void main(String[] args) {  
3   Rulletekst rulletekst = new Rulletekst("Heia Brann!");  
    // evt. bruke standardkonstruktør og setTekst() etterpå.  
3   rulletekst.start(); //Hvis Rulletekst extends Thread  
    new Thread(rulletekst).start(); //Hvis Rulletekst implements Runnable  
  
    try (Scanner kbd = new Scanner(System.in)) {  
2       while (true) {  
            System.out.print("Skriv inn ny tekst:");  
            String nyTekst = kbd.nextLine();  
2            rulletekst.setTekst(nyTekst);  
        }  
    }  
}
```

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. Nedenfor ser du 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 omdirigeres 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 for bokens forside

    @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

    ... konstruktører, gettere, settere, toString ...
}

@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;

    ... konstruktører, gettere, settere, toString ...
}
```

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 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 rett mot repo-ene.

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

- a) (10% ~ 24 min) 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.

Løsningsforslag:

```
1 @Service
  public class BokAnmeldelseService {

1     @Autowired private BokRepo bokRepo;
      @Autowired private AnmeldelseRepo anmeldelseRepo;

      public Bok finnBok(int id) {
2         return bokRepo.findById(id).orElse(null);
      }

      public void anmeldBok(
          int bokid, int rating, String tekst, String anmelder) {

1         Bok bok = finnBok(bokid);
2         Anmeldelse anmeldelse = new Anmeldelse(rating, tekst, anmelder, bok);
1         bok.leggTilAnmeldelse(anmeldelse);

2         anmeldelseRepo.save(anmeldelse);
      }
  }
```

- b) (10% ~ 24 min) 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.

Løsningsforslag:

Vi må anta at controlleren har en kobling til servicen fra a) som kan brukes til å hente aktuell bok, f.eks. slik:

```
@SController
public class BokAnmeldelseController {

    @Autowired private BokAnmeldelseService bas;

    ...
}
```



Da kan Get-mappingen gjøres slik:

```
2    @GetMapping("/anmelde")
    public String anmeldeskjema(
2        @RequestParam(name="bok_id") Integer id, Model model) {

1        if (id == null || bas.finnBok(id) == null) {
            return "feilmelding";
        }

1        Bok bok = bas.finnBok(id);
2        model.addAttribute("bok", bok);

2        return "anmelde";
    }
```

- c) (10% ~ 24 min) 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.

Løsningsforslag:

```
<html>
<body>
1   <a href="bokanmeldelser?bok_id=${bok.id}">Tilbake til anmeldelser</a><br>
1    ${bok.tittel}<br>

    Din anmeldelse:<br>
2   <form action="anmelde" method="post">
2       <input type="hidden" name="bok_id" value="${bok.id}" /><br>

    Rating<br>
1   <c:forEach var="i" begin="5" end="1" step="-1">
1       <input type="radio" name="rating" value="${i}" ${i eq 3 ? "checked" : ""}>
        <br>
    </c:forEach>

    Tekst<br>
    <textarea name="tekst" rows="5" cols="40"></textarea><br>
1   Hilsen <input type="text" name="anmelder"/><br>
1   <input type="submit" value="Send anmeldelse"><br>

    </form>

</body>
</html>
```

- d) (10% ~ 24 min) Skriv metoden i kontrolleren 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.

Løsningsforslag:

```
1  @PostMapping("/anmelde")
   public String anmelde(
2      @RequestParam(name="bok_id") int bokid,
      @RequestParam(name="rating") int rating,
      @RequestParam(name="tekst") String tekst,
      @RequestParam(name="anmelder") String anmelder,
      RedirectAttributes attributes) {

1      if (anmelder.isBlank()) {
          anmelder = "Anonym";
      }

2      bas.anmeldBok(bokid, rating, tekst, anmelder);

2      attributes.addAttribute("bok_id", bokid);
2      return "redirect:bokanmeldelser";
   }
```

- e) (10% ~ 24 min) 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 tilleggsfunksjonaliteten.

Løsningsforslag:

- b) *Her handler det om å få tak i cookien om den er med i requesten og legge den inn i modellen slik at den er tilgjengelig fra JSP-en.*

```
@GetMapping("/anmelde")
public String anmeldeskjema(
    @CookieValue(name="anmelder", required = false) String anmelder, ..){
    ...
    model.addAttribute("anmelder", anmelder);
    ...
}
```

- c) *Her handler det om å hente attributten fra modellen. I tillegg bør verdien escapes med <c:out> for å ufarliggjøre det å ha brukerinput direkte i nettsiden.*

```
<body>
...
Hilsen <input type="text" name="anmelder"
    value="<c:out value='${anmelder}' />" /><br>
...
</body>
```

- d) *Her handler det om å sende med anmelder som en cookie i responsen.*

```
@PostMapping("/anmelde")
public String anmelde(...,
    @RequestParam(name="anmelder") String anmelder,
    HttpServletResponse response) {
    ...
    Cookie c = new Cookie("anmelder", anmelder);
    response.addCookie(c);
    ...
}
```