

EKSAMENSOPPGAVE

Løsningsforslag

Emnekode: DAT108

Emnenavn: Programmering og webapplikasjoner

Klasse: 2. klasse DATA / INF

Dato: 10. juni 2024

Eksamensform: Skriftlig skoleeksamen (Wiseflow | FLOWmulti)

Eksamenstid: 4 timer (0900-1300)

Antall eksamensoppgaver: 6

Antall vedlegg: Ingen

Tillatte hjelpemiddel: Ingen

Oppgave 1 (10% ~ 24 minutter) – Strømmer

Flervalgsoppgave: Oppgavetekst i Wiseflow

Oppgave 2 (10% ~ 24 minutter) – Funksjonelle kontrakter

Flervalgsoppgave: Oppgavetekst i Wiseflow

Oppgave 3 (10% ~ 24 minutter) – JavaScript teori

Flervalgsoppgave: Oppgavetekst i Wiseflow

Se eget løsningsforslag for flervalgsoppgavene.

Oppgave 4 (15% ~ 36 minutter) – Tråder og trådsikkerhet

I denne oppgaven skal du jobbe med å simulere en iskremkiosk.

I iskremkiosken jobber det

- En selger som tar imot bestillinger fra kunder
- To iskremmakere som lager iskrem ut fra bestillingene
- To servitører som serverer ferdige iskremer til kundene

Kiosken holder orden på mottatte bestillinger og ferdige iskremer i to køer (én for bestillinger og én for ferdige iskremer).

Simuleringen gjøres ved at selger(e), iskremmaker(e) og servitør(er) legger inn og fjerner bestillinger og iskremer fra disse to køene i et visst tempo.

Koden vist nedenfor skisserer løsningen, men det mangler noe kode (som du skal legge inn), og løsningen har noen problemer (som du skal beskrive og løse).

```
class Selger implements Runnable {  
  
    private final Kiosk kiosk;  
  
    Selger(Kiosk kiosk) {  
        this.kiosk = kiosk;  
    }  
  
    @Override  
    public void run() {  
        while (true) {  
            kiosk.taImotBestilling();  
        }  
    }  
}
```

```
class Iskremmaker implements Runnable {  
  
    private final Kiosk kiosk;  
  
    Iskremmaker(Kiosk kiosk) {  
        this.kiosk = kiosk;  
    }  
  
    @Override  
    public void run() {  
        while (true) {  
            kiosk.lagIskrem();  
        }  
    }  
}
```

```
class Servitør implements Runnable {
```

```
    private final Kiosk kiosk;
```

```
    Servitør(Kiosk kiosk) {  
        this.kiosk = kiosk;  
    }
```

```
    @Override  
    public void run() {  
        while (true) {  
            kiosk.serverIskrem();  
        }  
    }  
}
```

```
class Kiosk {
```

```
    private final Queue<String> motatteBestillinger;  
    private final Queue<String> ferdigeIskremer;
```

```
    private int bestillingsTeller = 0;  
    private int iskremTeller = 0;
```

```
    Kiosk() {  
        motatteBestillinger = new LinkedList<>();  
        ferdigeIskremer = new LinkedList<>();  
    }
```

```
    // BESKRIVELSE:  
    // Simulerer at en selger tar imot en bestilling.
```

```
    public void taImotBestilling() {
```

```
        // Simulerer tiden det tar å ta imot en bestilling  
        Thread.sleep(500);
```

```
        // Øker telleren som teller antall bestillinger som er mottatt  
        bestillingsTeller++;
```

```
        String bestilling = "Bestilling " + bestillingsTeller;  
        System.out.println(bestilling);
```

```
        // Bestillingen er klar, og legges i kø for behandling  
        // API: Queue.offer(E e) - Legger inn elementet e bakerst i køen.  
        motatteBestillinger.offer(bestilling);  
    }
```

```

// BESKRIVELSE:
// Simulerer at en iskremmaker lager en is som er bestilt.

public void lagIskrem() {

    // En bestilling hentes ut, og fjernes fra køen
    // API: Queue.poll() - Henter ut og fjerner fremste element i køen,
    // eller returnerer null hvis køen er tom.
    String bestilling = motatteBestillinger.poll();

    // Simulerer tiden det tar å lage en is
    Thread.sleep(1000);

    // Øker telleren som teller antall is som har blitt lagd
    iskremTeller++;

    String iskrem = "Iskrem " + iskremTeller + " fra " + bestilling;
    System.out.println(iskrem);

    // Isen er nå klar til å serveres, og legges inn i iskremkøen
    ferdigeIskremer.offer(iskrem);
}

// BESKRIVELSE:
// Simulerer at en servitør serverer en iskrem.

public void serverIskrem() {

    // En is hentes ut, og fjernes fra køen
    String iskrem = ferdigeIskremer.poll();

    // Simulerer tiden det tar å servere en is
    Thread.sleep(300);

    // Isen er nå servert
    System.out.println("Serverer: " + iskrem);
}
}

```

```

public class IsMain {

    public static void main(String[] args) {

        Kiosk kiosk = new Kiosk();

        // a) Legg til kode for a) her. Se oppgaveteksten nedenfor.

        // b) Legg til kode for b) her. Se oppgaveteksten nedenfor.

    }
}

```

Oppgaver

- a) Skriv koden i main() som oppretter
- en tråd som simulerer en selger
 - to tråder som simulerer to iskremmakere
 - to tråder som simulerer to servitører

Løsningsforslag:

```
Thread selgerThread = new Thread(new Selger(kiosk));

Thread iskremmaker1Thread = new Thread(new Iskremmaker(kiosk));
Thread iskremmaker2Thread = new Thread(new Iskremmaker(kiosk));

Thread servitor1Thread = new Thread(new Servitør(kiosk));
Thread servitor2Thread = new Thread(new Servitør(kiosk));
```

- b) Skriv koden i main() som starter de 5 trådene du opprettet i a)

Løsningsforslag:

```
selgerThread.start();

iskremmaker1Thread.start();
iskremmaker2Thread.start();

servitor1Thread.start();
servitor2Thread.start();
```

- c) Forklar kort forskjellen på å skrive **selger.start()** og **selger.run()**.

Løsningsforslag:

- **selger.start()** starter en ny tråd hvor run-metoden i Runnable objektet blir kjørt. Dette gjør at koden i run-metoden kjører parallelt med hovedtråden og eventuelle andre tråder.
- **selger.run()** kaller run-metoden direkte, men i samme tråd som kallet ble gjort fra. Dette vil ikke skape noen parallellisme; det er rett og slett et vanlig metodekall og behandler koden sekvensielt.

- d) Klassen **Kiosk** bruker to køer (av typen Queue) implementert som kjedede lister (LinkedList) for å holde orden på mottatte bestillinger og iskrem som er klar til å serveres. Hvilke problemer kan vi få i simuleringen vår når vi velger å bruke Queue / LinkedList for køene? Hvordan kan vi fikse/forbedre koden slik at vi ikke får disse problemene? Du trenger ikke skrive kode, kun beskrive problemene og skissere mulige løsninger.

Løsningsforslag:

Problem:

- LinkedList er ikke trådsikker. Når flere tråder modifiserer listen samtidig (f.eks. to iskremmakere som legger til iskremer i køen), kan dette føre til inkonsistente tilstander eller konkurransetilstander.

Mulige løsninger:

1. Bruk av trådsikre implementasjoner som BlockingQueue
2. Synkroniseringsblokker/låsing rundt kritiske seksjoner

- e) Løsningen vår har også et annet problem, relatert til at vi har flere tråder som kjører samtidig. Gi en kort forklaring på hva som er galt, og beskriv kort hvordan dette kan fikses. Du trenger ikke skrive kode, kun beskrive problemene og skissere mulige løsninger. Hint: Se på tellerne.

Løsningsforslag:

Problem:

- Race Conditions: Tellerne bestillingsTeller og iskremTeller endres av forskjellige tråder, noe som kan føre til at to tråder leser og skriver til samme verdi samtidig.

Mulige løsninger:

- Atomic Variables: Bruk av atomiske variabler som AtomicInteger, som sikrer atomiske operasjoner for lesing og skriving.
- Synkronisering: Synkronisere metoder eller blokker som endrer eller leser disse tellerne.

- f) Gi en kort forklaring på hva en vranglås / dødlås (deadlock) er for noe. Du trenger ikke skrive kode.

Løsningsforslag:

- Definisjon: En vranglås eller dødlås oppstår når to eller flere tråder venter på hverandre for å frigjøre ressurser de holder på, slik at ingen av dem kan fortsette.

- Eksempel: Tenk deg to tråder som trenger to låser. Hvis hver tråd tar en lås og venter på den andre, er de fast i en evig ventesyklus.

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

En webside inneholder to felt. Det første feltet lar bruker registrere deltager i en konkurranse med sluttiden for deltager. Skjermbildet under viser hvordan dette feltet kan se ut:

Skjermbilde 1 viser et webform for registrering. Tittelen er "Deltager registrering". Det er to input-felter: et tekstfelt med innholdet "Anne Annesen" og et spinner-felt med verdien "23". Under disse feltene er det en bred knapp med teksten "Registrer resultat".

Skjermbilde 1: Felt for å registrere deltager

I Skjermbilde 1 blir deltager Anne Annesen registrert med en sluttid på 23 minutter. Deltager i dette skjermbildet kan representeres med et objekt på følgende form:

```
const deltager = {  
  'navn': 'Anne Annesen',  
  'tid': 23  
};
```

Kode-eksempel 1: Anne Annesen med tiden 23 minutter

Deltageren må lagres i en liste for senere bruk, f.eks. ved å bruke en forekomst av en JavaScript **Array**.

Det neste feltet lar bruker vise en deltager i konkurransen ved å søke på plassering. Skjermbildet under viser hvordan dette feltet kan se ut:

Skjermbilde 2 viser et webform for å vise resultater. Tittelen er "Resultater". Det er et spinner-felt med verdien "3" og en knapp med teksten "Vis deltager med plassering". Under disse er det en tekstlinje som sier "Anne Annesen har sluttid 23 minutter."

Skjermbilde 2: Resultat for deltager på tredje plass

I Skjermbilde 2 har bruker hentet fram deltager på tredje plass i konkurransen, som er Anne Annesen. Resultatlisten må altså være sortert med deltager som har korteste sluttid først.

Applikasjonen bruker HTML-koden under for å registre deltager:

```
<fieldset class="registrering" id="registrering">
  <legend>Deltager registrering</legend>

  <div>
    <input type="text" size="20" placeholder="Navn på deltager">
    <input type="number" size="11" placeholder="Tid i minutter">
    <button type="button">Registrer resultat</button>
  </div>
</fieldset>
```

Kode-eksempel 2: HTML-kode for å registrere deltager

Applikasjonen bruker HTML-koden under for å søke på deltager ut fra plassering i konkurransen:

```
<fieldset class="resultat" id="resultat">
  <legend>Resultater</legend>
  <div>
    <input type="number" size="3" value="1" min="1">
    <button>Vis deltager med plassering</button>
  </div>
  <p class="hidden plassering">
    <span></span> har sluttid <span></span> minutter.
  </p>
  <p class="plassholder">Her skal resultatet vises.</p>
</fieldset>
```

Kode-eksempel 3: HTML-kode for å søke på deltager

Applikasjonen inneholder følgende JavaScript-kode:

```
class KonkurranseKontroller {
  #liste = [];

  // Legg til her eventuelle flere private felt

  constructor(felt1ref, felt2ref) {
    this.#navnelement =
      felt1ref.querySelector("input[type='text']");
    this.#tidelement =
      felt1ref.querySelector("input[type='number']");
    const regbt = felt1ref.querySelector("button");
    regbt.addEventListener("click",
      () => { this.#regdeltager() }
    );

    // Legg inn kode her for felt2
  }

  #regdeltager() {
    // Legg inn kode her for å registrere deltager
  }

  // Legg til metod(er) for å kunne søke på deltager
}

const felt1 = document.getElementById("registrering");
const felt2 = document.getElementById("resultat");
new KonkurranseKontroller(felt1, felt2);
```

Kode-eksempel 4: JavaScript-kode for å registrere og søke på deltager

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

Hjelp:

- Kode-eksempelen under setter inn et nytt element i tabellen *this.#liste* mellom de eksisterende elementene med indeks 2 og indeks 3:

```
this.#liste.splice(3, 0, { 'navn': 'Anne Annesen', 'tid': 23 });
```

Det nye elementet får indeks 3 og alle påfølgende element øker sin indeks verdi med 1. Altså vil elementet som hadde indeks 3 få indeks 4.

- Kode-eksempelen under sorterer tabellen *this.#liste* stigende på sluttid til deltagerne:

```
this.#liste.sort((d1, d2) => { return d1.tid > d2.tid });
```

- Både **Map**, **Set** og **Array** har en metode *forEach*.
- HTML input elementer sin attributt *value* er nåværende verdi til feltet.

Løsningsforslag:

HTML-elementet med *class* lik plassholder skulle vært fjernet fra koden, og var tenkt å vises på websiden til et deltager-resultat ble vist. Dette elementet var ikke beskrevet i oppgaven, og funksjonaliteten til elementet er derfor også utelatt i løsningen.

For HTML-elementet med *class hidden* var tanken at dette først skulle vises når et resultat ble vist. Også dette ble fjernet fra oppgaven for å forenkle oppgaven, men overlevde i HTML-koden. Også dette er utelatt fra løsningsforslaget da det ikke er omtalt i oppgaveteksten.

```
class KonkurransKontroller {
  #liste = [];

  // Legg til her eventuelle flere private felt
  #navnelement;
  #tidelement;
  #plass;
  #reselement;

  constructor(felt1ref, felt2ref) {
    this.#navnelement = felt1ref.querySelector("input[type='text']");
    this.#tidelement = felt1ref.querySelector("input[type='number']");
    const regbt = felt1ref.querySelector("button");
    regbt.addEventListener("click", () => { this.#regdeltager() });

    // Legg inn kode her for felt2
    this.#plass = felt2ref.querySelector("input[type='number']");
    this.#reselement = felt2ref.querySelector("p.plassering");
    const plassbt = felt2ref.querySelector("button");
    plassbt.addEventListener("click", () => { this.#visresultat() });
  }
}
```

```

#regdeltager() {

    // Legg inn kode her for å registrere deltager

    /**
     * Metoden setter inn deltager på riktig plass i listen.
     * Listen er altså alltid sortert, og beste deltager
     * har index 0.
     */

    // Henter ut data for deltager fra websiden
    const navn = this.#navnelement.value;
    if (navn === "") {
        return;
    }
    const tidString = this.#tidelement.value;
    if (tidString == "") {
        return;
    }
    const tidInt = parseInt(tidString);

    // Finner riktig plassering i listen for deltager
    let index = 0;
    for (const e of this.#liste) {
        if (e.tid > tidInt) {
            break;
        }
        ++index;
    }

    /**
     * Nå vil "index" være riktig plass i listen for deltager.
     * Setter inn deltager i listen i posisjon "index".
     */
    this.#liste.splice(index, 0, { 'navn': navn, 'tid': tidInt });

    // Tømmer websiden for deltager-data fylt inn av bruker
    this.#navnelement.value = "";
    this.#tidelement.value = "";
}

```

```

// Legg til metod(er) for å kunne søke på deltager

#visresultat() {

    // Henter ut data fra websiden
    const plassString = this.#plass.value;
    if (plassString === "") {
        return;
    }
    const plassInt = parseInt(plassString);
    if (plassInt > this.#liste.length) {
        return;
    }

    const spanelms = this.#reselement.querySelectorAll("span");

    /**
     * Listen er fylt inn ferdig sortert. Henter ut deltager fra listen.
     * Må korrigere med "-1" da listen er 0-indeksert.
     */
    const deltager = this.#liste[plassInt - 1];
    spanelms[1].textContent = deltager.tid;
    spanelms[0].textContent = deltager.navn;
}

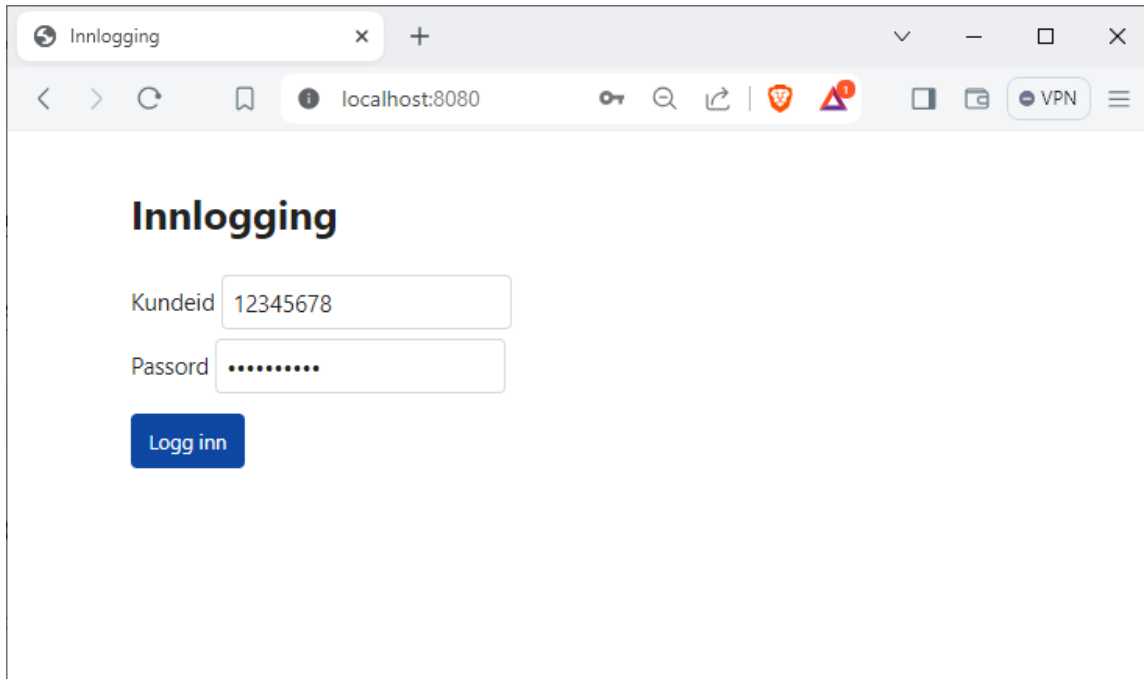
const felt1 = document.getElementById("registrering");
const felt2 = document.getElementById("resultat");
new KonkurransesKontroller(felt1,felt2);

```

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

I denne oppgaven skal vi jobbe litt med en kontooversikt i en tenkt nettbank.

Vi tenker oss at vi logger oss inn med kunde-id og passord som vist i dette skjermbildet:



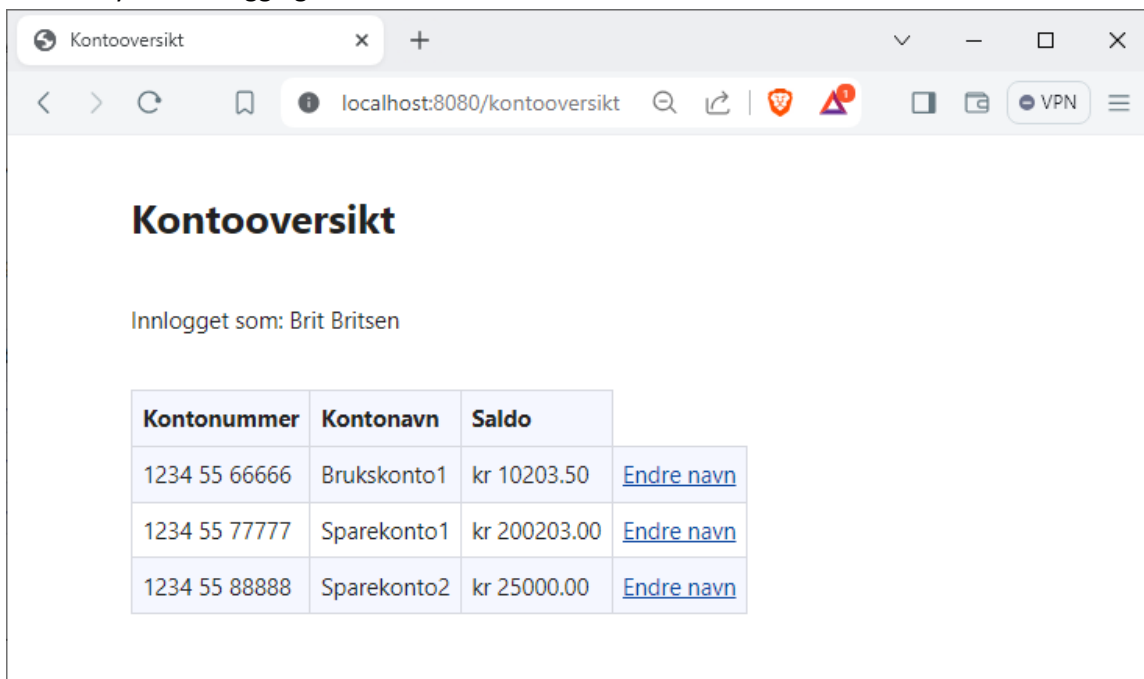
Innlogging

Kundeid

Passord

[Logg inn](#)

Etter vellykket innlogging kommer vi til kontooversikt:



Kontooversikt

Innlogget som: Brit Britsen

Kontonummer	Kontonavn	Saldo	
1234 55 66666	Brukskonto1	kr 10203.50	Endre navn
1234 55 77777	Sparekonto1	kr 200203.00	Endre navn
1234 55 88888	Sparekonto2	kr 25000.00	Endre navn

Vi ser at vi er innlogget som Brit Britsen, og at vi har 3 kontoer med kontonummer, kontonavn og saldo.

Vi ser også at det er en lenke (<a href ...>) for hver konto som gir oss mulighet til å endre kontonavn for denne kontoen.

Hvis vi klikker på lenken for den øverste kontoen i eksempelet, kommer vi til dette skjermbildet:

Endre kontonavn

Innlogget som: Brit Britsen

Kontonummer	Kontonavn
1234 55 66666	Brukskonto1

Lagre nytt navn

Her kan vi taste inn ønsket nytt kontonavn, trykke på knappen «Lagre nytt navn», og kontoen blir oppdatert med nytt navn i databasen. Etter oppdatering kommer vi tilbake til kontooversikten igjen.

Forespørsler om kontooversikt, endre-kontonavn-side, og lagring av nytt kontonavn krever at du er innlogget. For enkelthets skyld kan alle ugyldige forespørsler videresendes til en statisk html-side **feilside.html**.

Oversikt over request-mappings, views og redirects for gyldige/normale forespørsler:

Beskrivelse	Request	Tilhørende View	Neste
Se innloggingsside	GET /logginn	logginn.jsp	---
Logge inn	POST /logginn	---	/kontooversikt
Se Kontooversikt	GET /kontooversikt	kontooversikt.jsp	---
Se Endre kontonavn	GET /endrekontonavn	endrekontonavn.jsp	---
Lagre nytt kontonavn	POST /endrekontonavn	---	/kontooversikt

Litt om data og hjelpeklasser

Vi har lagret data om kunder og kontoer i en database, og henting og lagring av data gjøres via Spring-JpaRepositoriene **KundeRepo** og **KontoRepo**.

Java-entitetsklassen **Kunde** har disse egenskapene:

- int **id** (primærnøkkel)
- String **fornavn**
- String **etternavn**

Java-entitetsklassen **Konto** har disse egenskapene:

- String **kontonr** (primærnøkkel)
- String **kontonavn**
- BigDecimal **saldo**
- Kunde **kontoeier** (referanse til kunden som kontoen tilhører)

Du kan anta at disse klassene har de nødvendige get-metoder.

For enkelhets skyld tenker vi oss at alle requests blir håndtert av én Spring-controller kalt **BankController**.

For at BankController ikke skal trenge å forholde seg direkte til KundeRepo og KontoRepo, er det lagt inn en Spring-service **BankService** «oppå disse», som BankController kan bruke.

BankService har følgende metoder:

- Kunde **hentKunde**(int kundeid)
- Konto **hentKonto**(String kontonr)
- List<Konto> **hentAlleKontoerForKunde**(int kundeid)
- void **oppdaterKontoMedNyttNavn**(String kontonr, String nyttnavn)

Siden det er krav om innlogging, er det også laget en **LogginnService**, med følgende metoder:

- void **loggInn**(HttpServletRequest request, Kunde kunde)
- void **loggUt**(HttpServletRequest request)
- boolean **erLoggetInn**(HttpServletRequest request)
- Kunde **hentInnloggetKunde**(HttpServletRequest request)

Når bruker er innlogget vil et Kunde-objekt (uten liste av kontoer) for denne kunden være lagret som attributtet «kunde» i sesjonen.

Oppgaver

- a) Skriv en metode **henteKontooversikt(...)** i BankController, som håndterer forespørsel om å se Kontooversikt. Du tar utgangspunkt i at BankController er satt opp slik:

```
@Controller
public class BankController {

    @Autowired LogginnService logginnService;
    @Autowired BankService bankService;
    ...
}
```

Husk å sjekke at bruker er innlogget. Kontoene skal hentes på nytt fra databasen for hver forespørsel om å se kontooversikten (data kan være oppdatert siden sist).

Løsningsforslag:

```
2    @GetMapping("/kontooversikt")
    public String henteKontooversikt(HttpServletRequest request, Model model) {

1        if (!logginnService.erLoggetInn(request)) {
1            return "redirect:feilside.html";
        }

        //Trenger kunde.id for å hente kontoliste.
        //Kunde hentes fra sesjonsobjektet (v.hj.a. logginnService).
1        Kunde kunde = logginnService.hentInnloggetKunde(request);

2        List<Konto> kontoliste =
            bankService.hentAlleKontoerForKunde(kunde.getId());

2        model.addAttribute("kontoliste", kontoliste);

1        return "kontooversikt"; //jsp
    }
```

b) Skriv viewet for kontooversikten, **kontooversikt.jsp**. Du trenger kun å skrive <body>-delen.

Du blir ikke trukket om du ikke bruker tabell.

Du får poeng om du skriver teksten «Ingen aktive kontoer» i stedet for en tom tabell hvis innlogget kunde ikke har noen kontoer.

Løsningsforslag:

Forutsetninger:

- Sesjonen for innlogget kunde har en attributt "**kunde**", ref. oppgavetekst
- Model-objektet (fra oppg a) har fått lagt til attributten "**kontoliste**"

```
<body>
  <h3>Kontooversikt</h3>
  <p>Innlogget som: ${kunde.fornavn} ${kunde.etternavn}</p>
  1 xtra <c:if test="${!empty kontoliste}">
    <table>
      <tr><th>Kontonummer</th><th>Kontonavn</th><th>Saldo</th></tr>
      1 <c:forEach
      1   var="konto"           //Løkkevariabel
      1   items="${kontoliste}" //Samling det itereres over
      2   <tr><td>${konto.kontonrFormatert}</td>
        <td>${konto.kontonavn}</td>
        <td>kr ${konto.saldo}</td>
        <td><a href="
          1   endrekontonavn      //URL
          1   ?kontonr           //Parameternavn
          1   =${konto.kontonr}"> //Parameterverdi
            Endre navn</a></td>
        </tr>
      </c:forEach>
    </table>
  </c:if>
  1 xtra <c:if test="${empty kontoliste}">
    <p>Ingen aktive kontoer</p>
  </c:if>
</body>
```

- c) Skriv en metode **endreKontonavnSide(...)** i BankController, som håndterer forespørsel om å se endre-kontonavn-siden.

I tillegg til å sjekke om bruker er innlogget skal du her også sjekke at kunden eier den kontoen som skal endre navn. Tips: Lag gjerne en hjelpemetode for dette, f.eks. **boolean erGyldigKontoForInnloggetKunde(HttpServletRequest request, String kontonr)**. Denne kan også gjenbrukes i oppgave e).

Løsningsforslag:

Forutsetninger:

Requesten må inneholde en request-parameter (fra oppg b) som forteller hvilken konto som skal endre navn, f.eks. "**kontonr**"

```
1 @GetMapping("/endrekontonavn")
  public String endreKontonavnSide(HttpServletRequest request,
1      @RequestParam String kontonr, Model model) {
1      if ( !logginnService.erLoggetInn(request)
          || !erGyldigKontoForInnloggetKunde(request, kontonr)) {
          return "redirect:feilside.html";
      }
2      Konto konto = bankService.hentKonto(kontonr);
2      model.addAttribute("konto", konto);
1      return "endrekontonavn"; // .jsp
  }

  //Hjelpemetode
  private boolean erGyldigKontoForInnloggetKunde(
      HttpServletRequest request, String kontonr) {
1      Kunde kunde = logginnService.hentInnloggetKunde(request);
1      Konto konto = bankService.hentKonto(kontonr);
      return konto.getKontoeier().equals(kunde);
  }
```

- d) Skriv viewet for endre-kontonavn-siden, **endrekontonavn.jsp**. Du trenger kun å skrive <body>-delen.

Løsningsforslag:

Forutsetninger:

- Sesjonen for innlogget kunde har en attributt "**kunde**", ref. oppgavetekst
- Model-objektet (fra oppg c) har fått lagt til attributten "**konto**" for kontoen som vi skal gi nytt navn.

```
<body>
  <h3>Endre kontonavn</h3>
  <p>Innlogget som: ${kunde.fornavn} ${kunde.etternavn}</p>
  <form
    action="endrekontonavn"
    method="post">

    <table>
      <tr><th>Kontonummer</th><th>Kontonavn</th></tr>
      <tr><td>${konto.kontonrFormatert}</td>
      <td><input type="text"
        name="nyttnavn" value="${konto.kontonavn}" /></td></tr>
    </table>

    <input type="hidden" name="kontonr" value="${konto.kontonr}" />

    <input type="submit" value="Lagre nytt navn" />

  </form>
</body>
```

- e) Skriv en metode **lagreNyttKontonavn(...)** i BankController, som håndterer forespørsel om å endre navnet på en konto, altså foreta endringen i databasen.

I tillegg til å sjekke om bruker er innlogget skal du også her sjekke at kunden eier den kontoen som skal endre navn. Sjekk også at kontonavn er gyldig, f.eks. mellom 2-20 tegn.

Løsningsforslag:

Forutsetninger:

Requesten må inneholde to request-parametre (fra oppg d) som forteller hvilken konto som skal endre navn, og hva det nye navnet er, f.eks. parametre "kontonr" og "nyttnavn".

```
1 @PostMapping("/endrekontonavn")
2 public String lagreNyttKontonavn(HttpServletRequest request,
1 @RequestParam String kontonr, @RequestParam String nyttnavn) {
1     if (!logginnService.erLoggetInn(request)
        || !erGyldigKontoForInnloggetKunde(request, kontonr) //fra c)
        || !erGyldigKontonavn(nyttnavn)) { //se nedenfor
1         return "redirect:feilside.html";
        }
2     bankService.oppdaterKontoMedNyttNavn(kontonr, nyttnavn);
2     return "redirect:kontooversikt";
    }

    private boolean erGyldigKontonavn(String nyttnavn) {
        int navnlengde = nyttNavn.trim().size(); // Spring sjekker !null
1        return navnlengde >= 2 && navnlengde <= 20;
    }
```

- f) Skriv metoden **oppdaterKontoMedNyttNavn(...)** i BankService. Du tar utgangspunkt i at BankService er satt opp slik:

```
@Service
public class BankService {

    @Autowired KundeRepo kundeRepo;
    @Autowired KontoRepo kontoRepo;
    ...
}
```

Løsningsforslag:

```
public void oppdaterKontoMedNyttNavn(String kontonr, String nyttnavn) {

3       Konto konto = kontoRepo.findById(kontonr).orElse(null);
3       konto.setKontonavn(nyttnavn);

    // Et Spring repository bruker save() både ved «persist» og «merge»
    // Her blir det en «merge», siden kontoen finnes fra før.
4       kontoRepo.save(konto);
}
```