

## **EKSAMENSOPPGAVE**

**Emnekode: DAT108**

**Emnenavn: Programmering og webapplikasjoner**

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

**Dato: 7. desember 2020**

---

Eksamensform: Skriftlig hjemmeeksamen (Wiseflow | FLOWassign)

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

Antall eksamensoppgaver: 5

Antall sider (medregnet denne): 10

Antall vedlegg: Ingen

Tillatte hjelpemiddel: Alle.

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

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

**LYKKE TIL!**

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

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

- a) Gitt lambdauttrykket `s -> s.toLowerCase()`. Skriv en setning der du tilordner dette til en variabel. Husk å angi hvilken datatype variabelen er. Gi variabelen et passende navn. Tenk deg så at du har en liste av navn (av typen `List<String>`). Skriv kode, der variabelen over blir brukt, som gir utskrift med små bokstaver av alle navnene i listen, ett navn per linje.
- b) Gitt lambdauttrykket `tall -> tall % 3 == 0`. Skriv en metode med `x` som parameter som returnerer dette lambdauttrykket. Husk å få med returtype. Gi metoden et passende navn. Tenk deg så at du har en liste av heltall (av typen `List<Integer>`). Skriv kode, der metoden over blir brukt, som gir utskrift av alle tallene i listen som er delelige med 3, ett tall per linje.

Tenk at vi har en liste med passord i klartekst som vi ønsker å analysere styrken på, samt en liste med de 10 mest vanlige passordene:

```
List<String> passordlisten = Arrays.asList(
    "qwerty", "123", "password", "peace&love", "abc", "12345678", "admin",
    "tomee", "fotball", "hei på deg");

List<String> tiMestVanligePassord = Arrays.asList(
    "123456", "123456789", "qwerty", "password", "1234567", "12345678",
    "12345", "iloveyou", "111111", "123123");
```

- c) Bruk streams til å lage en ny liste med de passordene i **passordlisten** som også finnes i listen over **tiMestVanligePassord**. (Tips: `List` har en metode `contains()`).
- d) Bruk streams til å beregne hva som er gjennomsnittlig passordlengde for passordene i **passordlisten**. (Tips: Vær litt obs på dette med `Optional`).

## Oppgave 2 (8% ~ 20 minutter) – Passordsikkerhet

Anta at en angriper har full tilgang til passord-dataene, dvs. hash og salt for alle brukernes passord, og at hashing-algoritmen er kjent.

Vi snakker om tre ulike typer angrep man da kan gjøre for å finne brukeres passord.

1. Brute force-angrep
2. Ordlisteangrep
3. Tabelloppslag-angrep

En av sikkerhetsmekanismene vi bruker når vi hasher passord er **salting**. Hvilke(n) av de tre angrepstypene hjelper salting mot? Hvordan? Begrunn svaret. Kan salting også hjelpe mot en problematikk uavhengig av angrepsmåte?

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

- a) Nedenfor ser du JavaScript kode for to metoder som begge oppretter et nytt HTML P-element (tagg **P**) og legger inn noe data i P-elementet.

Metoden *showUserdata*:

```
function showUserdata(userData, rootElement) {
    rootElement.insertAdjacentHTML('beforeend', '<p>${userData}</p>`)
}
```

Metoden *viewUserdata*:

```
function viewUserdata(userData, rootElement) {
    rootElement.insertAdjacentHTML('beforeend', '<p></p>`)
    rootElement.lastElementChild.textContent = userData
}
```

- i. Gå gjennom koden linje for linje og forklar koden.
  - ii. Den ene av de to metoden over kan gi problemer i praksis. Diskuter dette og vis med et eksempel hvorfor denne metoden ikke bør brukes.
- b) Begge kodelinjene under vil opprette en liste av HTML-elementer med tagg **DIV**.
- ```
const divElementList = rootElement.querySelectorAll('div')
const divElements = rootElement.getElementsByTagName('div')
```
- Begge listene vil i utgangspunktet returnere de samme elementene, men vil kunne avvike etter hvert som JavaScript-kode arbeider med dokumentet.
- i. Utdyp forskjellen på de to listene, og vis eksempel på kode som gjør at de to listene ikke lengre inneholder de samme elementene.
  - ii. Legg til et HTML *class* attributt *info* på alle elementene i listen *divElementList*. Observer, elementene kan allerede ha andre *class* attributt elementer, og disse skal ikke røres.
- c) Opprett en JavaScript klasse **Calculator**. Klassen har en metode *calculate* og to egenskaper *status* og *result* som kan brukes utenfor klassen (brukes **public**).

Metode *calculate* er gitt av følgende kode:

```
/**
 * Metode for å utføre utregning
 * @public
 * @param {String} operation - Matematikk operasjon
 * @param {Array.<String>} numberList - Array av input-data
 */
calculate(operation, numberList) {
    /* JavaScript kode for å utføre utregning */
}
```

Parameter *operation* angir en regneoperasjon, og parameter *numberList* er en **Array** av heltall. Listen av heltall kan ha sitt opphav i HTML INPUT elementer. Hvert tall i listen kan derfor være en tekststreng som må konverteres til heltall før utregning.

Parameter *operation* kan ha en av følgende verdier:

- 'sum': Metoden skal finne summen av heltallene i *numberList*.
- 'produkt': Metoden skal finne produktet av heltallene i *numberList*.
- 'min': Metoden skal finne det minste tallet i *numberList*.
- 'max': Metoden skal finne det største tallet i *numberList*.

Metoden *calculate* skal gjøre svaret fra utregningen tilgjengelig i egenskapen *result*, og egenskapen *status* er en tekststreng med informasjon om utregningen.

Egenskapen *status* kan ha en av følgende verdier:

- 'Ingen tall i tallisten'
  - Angir at *numberList* ikke har noen elementer som er heltall.
- 'Tallisten inneholder verdi(er) som ikke er tall'
  - Angir at *numberList* har noen elementer som ikke er heltall. Disse har blitt ignorert i utregningen.
- 'Alle input-verdier ble prosessert'
  - Alle elementer i *numberList* er heltall og har blitt brukt i utregningen.

Eksempelet under viser bruk av **Calculator**:

```
const calculator = new Calculator()
calculator.calculate('sum', ['1','3','7'])
console.log(`Svaret er: ${calculator.result}`)
console.log(`Status for utregning: ${calculator.status}`)
```

Under vises utskriften i nettleserkonsollet fra koden over:

Svaret er: 11

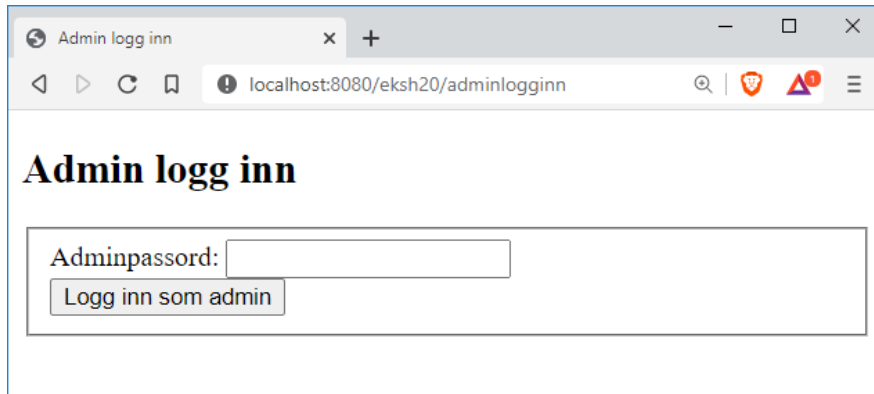
Status for utregning: Alle input-verdier ble prosessert

**Oppgave:** Skriv JavaScript koden for **Calculator** i samsvar med teksten over.

## Oppgave 4 (40% ~ 96 minutter) – Webapplikasjoner, tjenerside

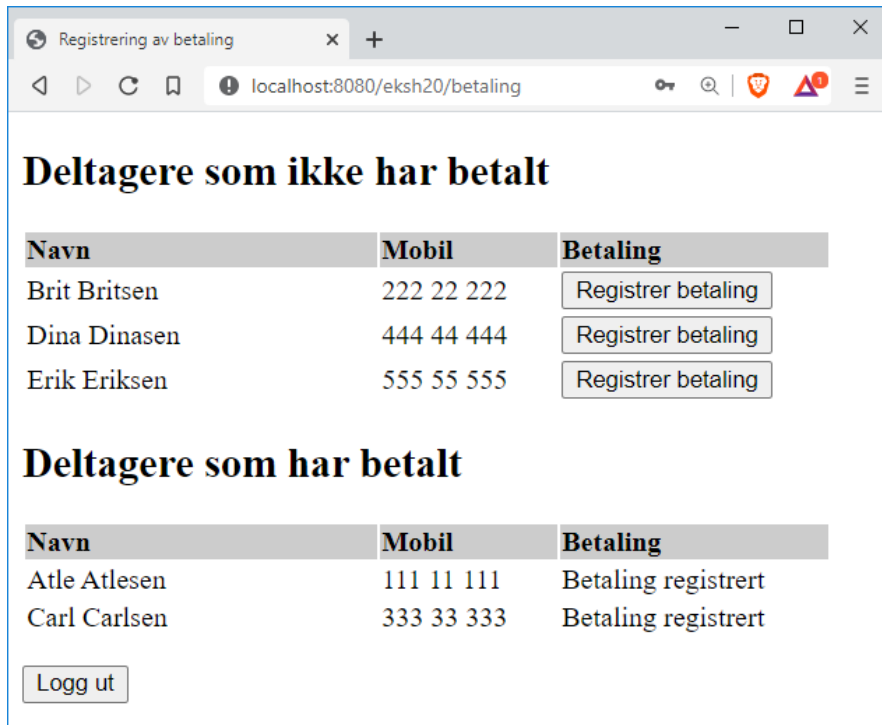
Vi skal se på en webapplikasjon der vi har en liste av deltagere påmeldt et arrangement. Arrangementet koster penger, og vi vil lage en løsning for å registrere og holde oversikt over innkomne betalinger.

Man må være innlogget som en slags "admin" for å kunne se betalingsoversikt og for å kunne registrere betalinger. Man logger seg inn som "admin" via et hardkodet hemmelig passord:



figur 4.1

Etter vellykket admin-innlogging kommer man til betalingsoversikten, som ser slik ut:



figur 4.2

Når man trykker på knappen "Registrer betaling" blir betaling registrert i databasen for aktuell deltager og man får opp betalingsoversikten på nytt.

F.eks. hvis man trykker på knappen for Dina Dinassen i figur 4.2 vil resultatsiden se slik ut:

Registrering av betaling

localhost:8080/eksh20/betaling

### Deltagere som ikke har betalt

| Navn         | Mobil      | Betaling           |
|--------------|------------|--------------------|
| Brit Britsen | 222 22 222 | Registrer betaling |
| Erik Eriksen | 555 55 555 | Registrer betaling |

### Deltagere som har betalt

| Navn          | Mobil      | Betaling            |
|---------------|------------|---------------------|
| Atle Atlesen  | 111 11 111 | Betaling registrert |
| Carl Carlsen  | 333 33 333 | Betaling registrert |
| Dina Dinassen | 444 44 444 | Betaling registrert |

Logg ut

figur 4.3

Hvis man prøver å se betalingsoversikten eller prøver å registrere betaling uten å være innlogget som admin skal man omdirigeres til admin-logginn med melding om at innlogging er påkrevd, slik:

Admin logg inn

Forespørselen din krever pålogging som admin

Adminpassord:

Logg inn som admin

figur 4.4

Utlogging fra betalingsoversikten ("Logg ut"-knappen) eller feil adminpassord ved pålogging skal også gi en omdirigering til admin logginn-siden.

Java-klasser og JSP-sider som skal være med i løsningen:

**/adminlogginn** mappes til **AdminLogginnServlet.java**

- `init()` har ansvar for å hente inn "adminpassord" fra `web.xml` (en `init-parameter`) og lagre det i en instansvariabel for senere oppslag.
- `doGet()` har ansvar for å få opp logginn-skjema med evt. feilmeldinger
  - ... i samarbeid med **AdminLogginn.jsp**
- `doPost()` har ansvar for å logge brukeren inn som admin

**/betaling** mappes til **BetalingServlet.java**

- `doGet()` har ansvar for å få opp betalingsoversikten
  - ... i samarbeid med **Betaling.jsp**
- `doPost()` har ansvar for å få registrert en betaling

**/adminloggut** mappes til **AdminLoggutServlet.java**

- `doPost()` har ansvar for å logge ut admin-brukeren

**@Entity** public class **Deltager** //JPA-entitet for deltagerne som er lagret i databasen

```
@Id private String mobil
private String navn
private boolean betalt
... gettere og settere
```

**DeltagerDAO.java** //Hjelpeklasse for databaseoperasjoner

- `List<Deltager> finnAlle()` //Henter ut en liste over alle registrerte deltagere
- `void registrerBetalingFor(mobil)` //Registrerer betaling for deltager med gitt mobil
- ...

**InnloggingUtil.java** //Hjelpeklasse for adgangskontroll, innlogging og utlogging

- `static void adminLogginn(request)` //Logger bruker i denne sesjonen inn som admin
- `static void adminLoggut(request)` //Logger bruker i denne sesjonen ut som admin
- `static boolean isInnloggetSomAdmin(request)` //Om bruker er innlogget som admin
- ...

Oppgaven deres er å implementere deler av denne løsningen. Det dere ikke skal implementere selv kan dere anta finnes og virker.



## Krav til løsningen

For å få full score må du løse oppgaven på best mulig måte i hht. prinsippene i kurset, dvs. Model-View-Controller, Post-Redirect-Get, EL og JSTL i JSP-ene, trådsikkerhet, robusthet, unntakshåndtering, god bruk av hjelpeklassene, elegant kode, osv ...

Dere trenger ikke ha med import-setninger eller andre direktiver i løsningene deres.

Opgavene:

a) Skriv hele **AdminLogginnServlet.java**

b) Skriv **doGet()** i **BetalingServlet.java**

c) Skriv **Betaling.jsp**

**Tips:** Siden inneholder mange "Registrer betaling"-knapper. En grei måte å skille disse fra hverandre er å ha én <form>-tag per knapp.

d) Skriv **doPost()** i **BetalingServlet.java**

e) Skriv **adminLogginn(request)** i **InnloggingUtil.java**

f) Skriv **registrerBetalingFor(mobil)** i **DeltagerDAO.java**

## Oppgave 5 (16% ~ 38 minutter) – Tråder

- a) Vi ønsker å legge inn en tråd i et program som gjør "bitcoin-mining" i bakgrunnen.

"Bitcoin-mining" i denne sammenhengen betyr helt enkelt å gå gjennom en for-løkke for alle positive (long) heltall, beregne en hash for hvert enkelt tall, og sjekke om verdien av hashen starter med fem nullere, altså `.startsWith("00000")`. I så fall skal tallet og hashen skrives ut på skjermen. F.eks. slik:

`262997966 -> 00000PM477MtJDRsn+9RjMhPooy/12F4Vj7a+WbbZ30=`

Du kan anta at du har en metode `static String hashOf(long number)` som utfører selve hashingen av tallet.

Tråden skal fortsette beregningene til alle tall t.o.m. `Long.MAX_VALUE` er gjennomløpt.

Skriv en **main-metode** som oppretter og starter en tråd som gjør slik "bitcoin-mining". (Vi får innbille oss at main også gjør andre ting etterpå, men det tenker vi ikke på her). TIPS: Lag en **hjelpemetode** for selve logikken som skal utføres i tråden slik at `run()` kun trenger å kalle denne metoden.

- b) Vi skal lage et lite program der flere tråder disponerer en felles "bankkonto". Bankkontoen skal blokkere mot overtrekk, dvs. at tråder som ønsker å gjøre uttak som vil føre til negativ saldo må vente til det er satt inn nok penger før uttaket blir gjennomført.

Skriv en **main-metode** der det opprettes et Bankkonto-objekt og to tråder som gjør en del innskudd og uttak på denne bankkontoen. Skriv også **Bankkonto**-klassen. (Du trenger ikke å ha med kode for unntakshåndtering.)

# **EKSAMENSOPPGÅVE**

**Emnekode: DAT108**

**Emnenamn: Programmering og webapplikasjoner**

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

**Dato: 7. desember 2020**

---

Eksamensform: Skriftleg hjemmeeksamen (Wiseflow | FLOWassign)

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

Tal på eksamensoppgaver: 5

Tal på sider (medrekna denne): 10

Tal på vedlegg: Ingen

Tillatne hjelpemiddel: Alle

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

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

**LUKKE TIL!**

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

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

- Gitt lambdauttrykket `s -> s.toLowerCase()`. Skriv ei setning der du tilordnar dette til ein variabel. Hugs å angi kva datatype variabelen er. Gi variabelen eit passende namn. Tenk deg så at du har ei liste av namn (av typen `List<String>`). Skriv kode, der variabelen over blir brukt, som gir utskrift med små bokstavar av alle namna i lista, eitt namn per linje.
- Gitt lambdauttrykket `tall -> tall % x == 0`. Skriv ein metode med `x` som parameter som returnerer dette lambdauttrykket. Hugs å få med returtype. Gi metoden eit passende namn. Tenk deg så at du har ei liste av heiltal (av typen `List<Integer>`). Skriv kode, der metoden over blir brukt, som gir utskrift av alle tala i listen som er delelige med 3, eitt tal per linje.

Tenk at vi har ei liste med passord i klartekst som vi ønsker å analysere styrken på, og dessutan ei liste med dei 10 mest vanlege passorda:

```
List<String> passordlisten = Arrays.asList(
    "qwerty", "123", "password", "peace&love", "abc", "12345678", "admin",
    "tomee", "fotball", "hei på deg");

List<String> tiMestVanligePassord = Arrays.asList(
    "123456", "123456789", "qwerty", "password", "1234567", "12345678",
    "12345", "iloveyou", "111111", "123123");
```

- Bruk streams til å laga ei ny liste med dei passorda i `passordlisten` som også finst i lista over `tiMestVanligePassord`. (Tips: List har ein metode `contains()`).
- Bruk streams til å berekne kva som er gjennomsnittleg passordlengde for passorda i `passordlisten`. (Tips: Vær litt obs på dette med `Optional`).

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

Anta at ein angripar har full tilgang til passord-dataa, dvs. hash og salt for passordet til alle brukarane, og at hashing-algoritmen er kjent.

Vi snakkar om tre ulike typar angrep ein då kan gjera for å finna passordet til brukarar.

1. Brute force-angrep
2. Ordlisteangrep
3. Tabelloppslag-angrep

Ein av tryggingmekanismene vi bruker når vi hasher passord er **salting**. Kva (bokmål: Hvilke(n)) av dei tre angrepstypene hjelper salting mot? Korleis? Grunngi svaret. Kan salting òg hjelpa mot ein problematikk uavhengig av angrepsmåte?

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

- a) Nedanfor kan du sjå JavaScript kode for to metodar som begge opprettar eit nytt HTML P-element (tagg **P**) og legg inn nokre data i P-elementet.

Metoden *showUserdata*:

```
function showUserdata(userData, rootElement) {
    rootElement.insertAdjacentHTML('beforeend', '<p>${userData}</p>`)
}
```

Metoden *viewUserdata*:

```
function viewUserdata(userData, rootElement) {
    rootElement.insertAdjacentHTML('beforeend', '<p></p>`)
    rootElement.lastElementChild.textContent = userData
}
```

- i. Gå gjennom koden linje for linje og forklar koden.
  - ii. Den eine av dei to metodene over kan gje problem i praksis. Diskuter dette og vis med eit døme kvifor denne metoden ikkje bør verte nytta.
- b) Begge kodelinene nedanfor vil opprette ei liste av HTML-element med tagg **DIV**.
- ```
const divElementList = rootElement.querySelectorAll('div')
const divElements = rootElement.getElementsByTagName('div')
```
- Begge lista vil i utgangspunktet returnere dei same elementa, men vil kunne avvike etterkvart som JavaScript-kode arbeider med dokumentet.
- i. Klargjer forskjellen på dei to lista, og vis døme på kode som gjer at dei to lista ikkje lengre inneheld dei same elementa.
  - ii. Legg til eit HTML *class* attributt *info* på alle elementa i lista *divElementList*. Observer, elementa kan allereie ha andre *class* attributt element, og disse skal ikkje blir endra.
- c) Opprett ein JavaScript klasse **Calculator**. Klassen har ein metode *calculate* og to eigenskapar *status* og *result* som kan verte nytta utanfor klassen (kan verte nytta **public**).

Metode *calculate* er gitt av fylgjande kode:

```
/**
 * Metode for å utføre utrekninga
 * @public
 * @param {String} operation - Matematikk operasjon
 * @param {Array.<String>} numberList - Array av input-data
 */
calculate(operation, numberList) {
    /* JavaScript kode for å utføre utrekning */
}
```

Parameter *operation* angir ein rekneoperasjon, og parameter *numberList* er ein **Array** av heiltal. Lista av heiltal kan ha sitt opphav i HTML INPUT element. Kvart tall i lista kan derfor være ein tekststreng som må verte konvertert til heiltal før utrekning.

Parameter *operation* kan ha en av følgende verdi:

- 'sum': Metoden skal rekne ut summen av heiltala i *numberList*.
- 'produkt': Metoden skal rekne ut produktet av heiltalla i *numberList*.
- 'min': Metoden skal finne det minste talet i *numberList*.
- 'max': Metoden skal finne det største talet i *numberList*.

Metoden *calculate* skal gjere svaret frå utrekninga tilgjengeleg i eigenskapen *result*, og eigenskapen *status* er ein tekststreng med informasjon om utrekninga.

Eigenskapen *status* kan ha ein av følgende verdi:

- 'Ingen tall i tallista'
  - Fortel at *numberList* ikkje har nokre element som er heiltal.
- 'Tallista inneheld verdi som ikkje er tal'
  - Angir at *numberList* har nokre element som ikkje er heiltal. Disse har blitt ignorert i utrekninga.
- 'Alle input-verde blei prosessert'
  - Alle elementa i *numberList* er heiltal og har verte nytta i utrekninga.

Dømet under viser bruk av **Calculator**:

```
const calculator = new Calculator()
calculator.calculate('sum', ['1','3','7'])
console.log(`Svaret er: ${calculator.result}`)
console.log(`Status for utrekninga: ${calculator.status}`)
```

Nedanfor kan du sjå utskrifta i nettlesarkonsollet frå koden over:

Svaret er: 11

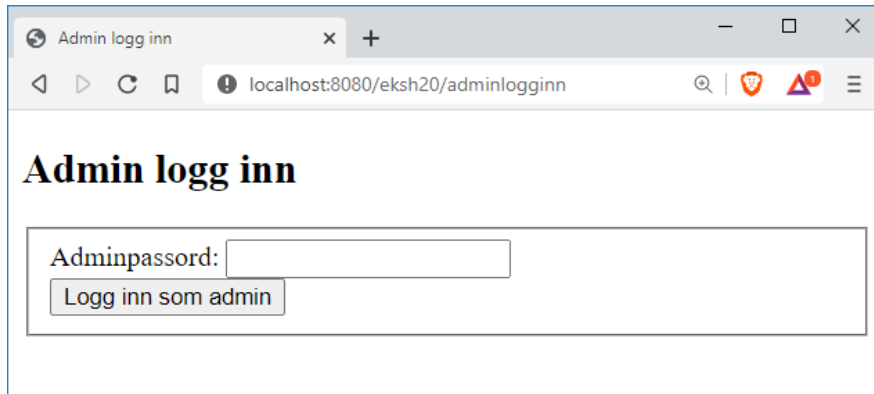
Status for utrekninga: Alle input-verde blei prosessert

**Oppgave:** Skriv JavaScript koden for **Calculator** i samsvar med teksten over.

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

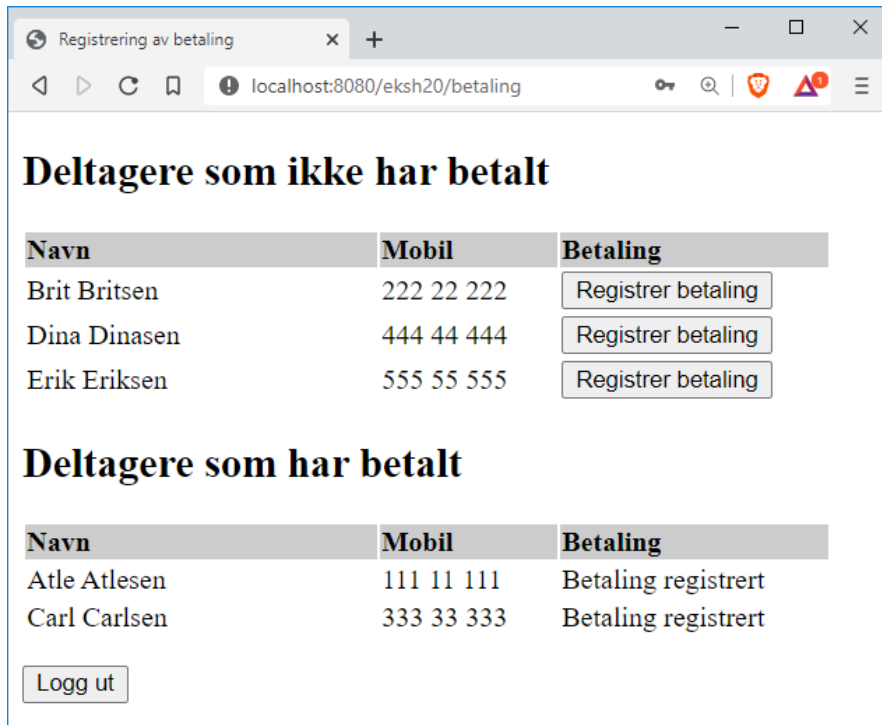
Vi skal sjå på ein webapplikasjon der vi har ei liste av deltakarar påmeldt eit arrangement. Arrangementet kostar pengar, og vi vil laga ei løysing for å registrera og halda oversikt over innkomne betalningar.

Ein må vera innlogga som ein slags "admin" for å kunna sjå betalingsoversikt og for å kunna registrera betalningar. Ein loggar seg inn som "admin" via eit hardkodet hemmeleg passord:



figur 4.1

Etter vellykka admin-innlogging kjem ein til betalingsoversikta, som ser slik ut:



Navn	Mobil	Betaling
Brit Britsen	222 22 222	Registrer betaling
Dina Dinassen	444 44 444	Registrer betaling
Erik Eriksen	555 55 555	Registrer betaling

Navn	Mobil	Betaling
Atle Atlesen	111 11 111	Betaling registrert
Carl Carlsen	333 33 333	Betaling registrert

figur 4.2

Når ein trykker på knappen "Registrer betaling" blir betaling registrert i databasen for aktuell deltakar og ein får opp betalingsoversikta på nytt.



T.d. viss ein trykker på knappen for Dina Dinassen i figur 4.2 vil resultatsida sjå slik ut:

Registrering av betaling

localhost:8080/eksh20/betaling

### Deltagere som ikke har betalt

Navn	Mobil	Betaling
Brit Britsen	222 22 222	Registrer betaling
Erik Eriksen	555 55 555	Registrer betaling

### Deltagere som har betalt

Navn	Mobil	Betaling
Atle Atlesen	111 11 111	Betaling registrert
Carl Carlsen	333 33 333	Betaling registrert
Dina Dinassen	444 44 444	Betaling registrert

Logg ut

figur 4.3

Viss ein prøver å sjå betalingsoversikta eller prøver å registrera betaling utan å vera innlogga som admin skal ein omdirigerast til admin-logginn med melding om at innlogging er påkravde, slik:

Admin logg inn

Forespørselen din krever pålogging som admin

Adminpassord:

Logg inn som admin

figur 4.4

Utlogging frå betalingsoversikta ("Logg ut"-knappen) eller feil adminpassord ved pålogging skal òg gi ein omdirigering til admin logginn-siden.

Java-klasser og JSP-sider som skal være med i løsningen:

**/adminlogginn** mappes til **AdminLogginnServlet.java**

- `init()` har ansvar for å hente inn "adminpassord" fra `web.xml` (en `init-parameter`) og lagre det i en instansvariabel for seinare oppslag.
- `doGet()` har ansvar for å få opp logginn-skjema med evt. feilmeldingar
  - ... i samarbeid med **AdminLogginn.jsp**
- `doPost()` har ansvar for å logge brukaren inn som admin

**/betaling** mappes til **BetalingServlet.java**

- `doGet()` har ansvar for å få opp betalingsoversikten
  - ... i samarbeid med **Betaling.jsp**
- `doPost()` har ansvar for å få registrert en betaling

**/adminloggut** mappes til **AdminLoggutServlet.java**

- `doPost()` har ansvar for å logge ut admin-brukeren

**@Entity** public class **Deltager** //JPA-entitet for deltagarane som er lagra i databasen

```
@Id private String mobil
private String navn
private boolean betalt
... gettere og settere
```

**DeltagerDAO.java** //Hjelpeklasse for databaseoperasjoner

- `List<Deltager> finnAlle()` //Hentar ut en liste over alle registrerte deltagarar
- `void registrerBetalingFor(mobil)` //Registrerer betaling for deltagar med gitt mobil
- ...

**InnloggingUtil.java** //Hjelpeklasse for adgangskontroll, innlogging og utlogging

- `static void adminLogginn(request)` //Logger bruker i denne sesjonen inn som admin
- `static void adminLoggut(request)` //Logger bruker i denne sesjonen ut som admin
- `static boolean isInnloggetSomAdmin(request)` //Om bruker er innlogga som admin
- ...

Oppgåva dykkar er å implementera delar av denne løysinga. Det de ikkje skal implementera sjølv kan de anta finst og verkar.

## Krav til løysinga

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

De treng ikkje ha med import-setningar eller andre direktiv i løysingane dykkar.

Oppgåvene:

a) Skriv heile **AdminLogginnServlet.java**

b) Skriv **doGet()** i **BetalingServlet.java**

c) Skriv **Betaling.jsp**

**Tips:** Sida inneheld mange "Registrer betaling"-knappar. Ein grei måte å skilja desse frå kvarandre er å ha éin <form>-tag per knapp.

d) Skriv **doPost()** i **BetalingServlet.java**

e) Skriv **adminLogginn(request)** i **InnloggingUtil.java**

f) Skriv **registrerBetalingFor(mobil)** i **DeltagerDAO.java**

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

- a) Vi ønsker å leggja inn ein tråd i eit program som gjer "bitcoin-mining" i bakgrunnen.

"Bitcoin-mining" i denne samanhengen betyr heilt enkelt å gå gjennom ein for-løkke for alle positive (long) heiltal, berekna ein hash for kvart enkelt tal, og sjekka om verdien av hashen startar med fem nullere, altså `.startsWith("00000")`. I så fall skal talet og hashen bli skrive ut på skjermen. T.d. slik:

`262997966 -> 00000PM477MtJDRsn+9RjMhPooy/12F4Vj7a+WbbZ30=`

Du kan anta at du har ein metode `static String hashOf(long number)` som utfører sjølve hashing av talet.

Tråden skal halda fram med berekningane til alle tal t.o.m. `Long.MAX_VALUE` er gjennomløpt.

Skriv ein **main-metode** som opprettar og startar ein tråd som gjer slik "bitcoin-mining". (Vi får innbilla oss at main òg gjør andre ting etterpå, men det tenker vi ikkje på her). TIPS: Lag ein **hjelpemetode** for sjølve logikken som skal utførast i tråden slik at `run()` berre treng å kalla denne metoden.

- b) Vi skal laga eit lite program der fleire trådar disponerer ein felles "bankkonto". Bankkontoen skal blokkera mot overtrekk, dvs. at trådar som ønsker å gjera uttak som vil føra til negativ saldo må venta til det er sett inn nok pengar før uttaket blir gjennomført.

Skriv ein **main-metode** der det blir oppretta eit Bankkonto-objekt og to trådar som gjer ein del innskott og uttak på denne bankkontoen. Skriv òg **Bankkonto**-klassen. (Du treng ikkje å ha med kode for unntakshandtering.)