

EKSAMENSOPPGAVE

Emnekode: DAT108

Emnenavn: Programmering og webapplikasjoner

Klasse: 2. klasse DATA / INF

Dato: 8. juni 2022

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 (24% ~ 58 minutter) – Lambda-uttrykk og strømmer

Generelt for alle spørsmål i Oppgave 1: For å få god uttelling må løsningene ikke være unødig kompliserte, og det må brukes lambdauttrykk, metodereferanser og streams der det passer.

a) Hva er datatypene til følgende uttrykk:

- i. `a -> a.length()`
- ii. `(Integer a) -> String.valueOf(a)`
- iii. `a -> a.getDayOfWeek().equals(DayOfWeek.MONDAY)`

Om mulig, vis også hvordan uttrykkene over kan omskrives til metodereferanser.

b) Anta at vi har en **liste av Person-objekter** referert av liste-variabelen **venner**, og vi ønsker å få skrevet dem ut sortert på et par ulike måter. Person-klassen ser du nedenfor.

```
public class Person {  
    String fulltNavn;  
    LocalDate foddato;  
  
    // + konstruktører, gettere, settere og toString  
}
```

- i. Skriv en main-metode som skriver listen ut sortert på navn, en person per linje.
- ii. Skriv en main-metode som skriver listen ut sortert etter når på året de har bursdag (året betyr ikke noe, kun når på året) TIPS: Siden året ikke skal være med i sammenligning kan dette settes til et fast år i sammenligningen ved å bruke `LocalDate.withYear()`-metoden.

- c) Denne oppgaven går ut på å parse og summere sammen alle positive heltall i en tekst.

Vi kan f.eks. ha teksten "Per 6 32 Anne xyz Bergen 14" som skal gi svaret 52 (altså 6+32+14).

Det første steget kan være å dele opp teksten i tokens med `t.split(" ")`.

Deretter er det å plukke ut det som er tall. Til dette har jeg definert et predikat som sier om en tekst er et heltall. Dette forventes brukt i løsningen.

```
Predicate<String> erEtPositivtHeltall = t -> t.matches("^\\d+$");
```

Lag en `main()`-metode som tar utgangspunkt i en tekst (lagret i en `String`-variabel), og som ved hjelp av tipsene over beregner summen av alle positive heltall i teksten. Bruk `streams-API`et. Svaret skal til slutt skrives ut på skjermen.

- d) Vi har laget en metode for å skrive ut en tabell av funksjonsverdier for kvadratfunksjonen $f(x) = x^2$:

```
public class Printer {
    public static void printTabellForKvadratfunksjon(
        double start, double stopp, double steg) {

        for (double x=start; x<=stopp; x+=steg) {
            double fx = x*x;
            System.out.printf("%8.3f%8.3f\n", x, fx);
        }
    }
}
```

Bruken av denne kan f.eks. være slik (med resultat til høyre):

```
public static void main(String... blablabla) {
    Printer.printTabellForKvadratfunksjon(1, 5, 1);
}
```

1,000	1,000
2,000	4,000
3,000	9,000
4,000	16,000
5,000	25,000

Vi ønsker å lage en mer generell metode som skriver ut en tabell av funksjonsverdier for en vilkårlig funksjon (f.eks. $f(x) = \sin(x)$, $f(x) = \log(x)$, osv..).

- Lag en metode tilsvarende den som er vist over slik at dette er mulig. Kall den nye metoden `printTabellForFunksjon(...)`.
- Bruk den nye generelle metoden fra i. i `main`-programmet i stedet for den gamle til å skrive ut kvadrattallene fra 1 til 5 som i eksemplet.

Oppgave 2 (24% ~ 58 minutter) – JavaScript

- a) En applikasjon for en stoppeklokke inneholder JavaScript-koden under:

```
class TimerController {  
    #start = Date.now();  
  
    constructor(container) {  
        const bt = container.querySelector("button");  
        bt.addEventListener("click", this.runde);  
    }  
  
    runde() {  
        console.log(Date.now() - this.#start);  
    }  
}
```

Applikasjonen har en knapp for å vise rundetider i web-konsollet.

Du kan anta at den tilhørende HTML-koden er riktig laget og inkluderer JavaScript-koden på riktig måte. Det er heller ingen syntaksfeil i JavaScript-koden.

- Konstruktøren inneholder en feil som gjør at applikasjonen ikke kan virke. Utdyp hva problemet er.
 - Omskriv koden og fjern feilen ved å bruke **Function** sin metode *bind*.
 - Problemet kan også løses ved å kalle *runde* i en omsluttende funksjon. Utdyp hvorfor et funksjonsuttrykk med pil-notasjon kan løse problemet, mens et standard funksjonsuttrykk kan gi problemer.
- b) Opprett en JavaScript-klasse **Ordanalyse**. Instanser av klassen skal kunne analysere en tekst, der du kun skal implementere tre metoder, *setTekst*, *getAntallord* og *getOrdliste*.

Metoden *setTekst* brukes for å registrere teksten som skal analyseres, metoden *getAntallord* skal returnere antall unike ord i teksten, og metoden *getOrdliste* skal returnere en liste av alle unike ord i teksten.

Ordanalyse skal ikke skille mellom store og små bokstaver når unike ord registreres eller telles, og listen med ord som returneres av *getOrdliste* skal kun bruke små bokstaver.

Kodeeksempelet under viser bruk av klassen **Ordanalyse**:

```
const oa = new Ordanalyse();  
oa.setTekst("Velkommen! Nå, ja; ja nå er det eksamenstid.");  
const antall = oa.getAntallord();  
const tabell = oa.getOrdliste();
```

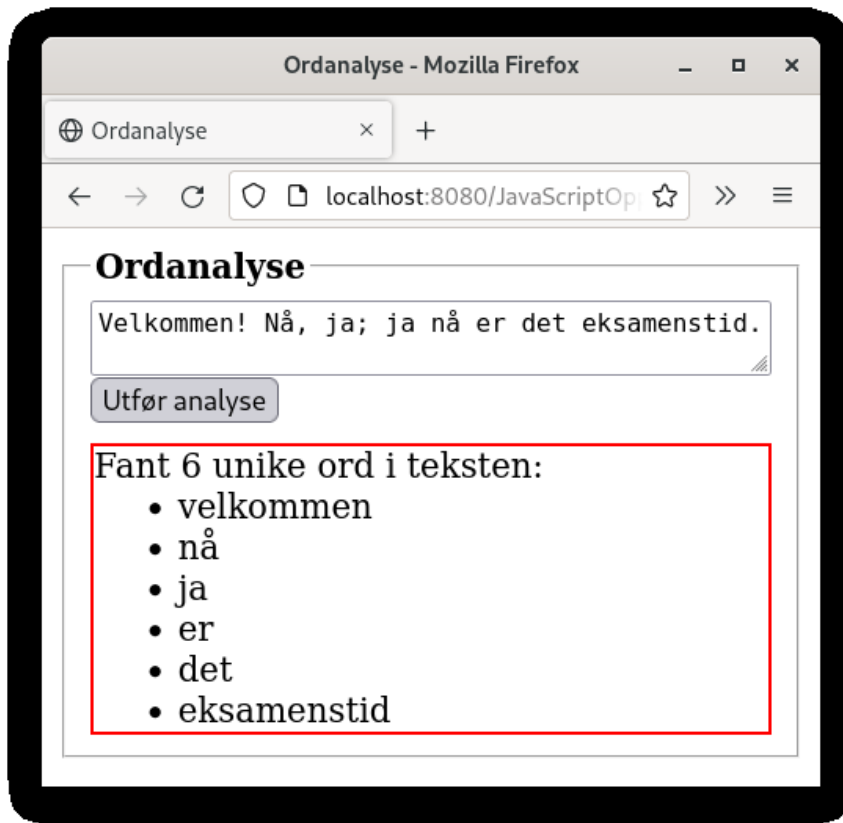
Variabelen *antall* vil nå ha verdien 6, og variabelen *tabell* skal ha følgende innhold:

```
["velkommen", "nå", "ja", "er", "det", "eksamenstid"]
```

En løsning for å registrere unike ord er å bruke ordet som nøkkel (key) i en JavaScript **Map**.

Oppgave: Skriv JavaScript-kode for **Ordanalyse** med metodene *setTekst*, *getAntallord* og *getOrdliste* i samsvar med teksten over.

- c) Figuren nedenfor viser en webside som bruker JavaScript-klassen **Ordanalyse** fra forrige oppgave:



Figur 1: Webside for ordanalyse

Form-elementet som vises i figuren over er gitt av følgende HTML-kode:

```
<form>
  <fieldset>
    <legend>Ordanalyse</legend>
    <textarea placeholder="Fyll inn tekst" data-tekst></textarea>
    <button type="button" data-analyser>Utfør analyse</button>
    <div data-resultat></div>
  </fieldset>
</form>
```

Ved klikk på knappen «Utfør analyse», eller når websiden lastes på nytt skal JavaScript-kode produsere innholdet som er vist inne i den røde rammen i figuren.

Oppgave: Skriv JavaScript-kode for å implementer funksjonaliteten som er beskrevet over. Løsningen må bruke JavaScript-klassen **Ordanalyse** fra forrige oppgave.

Oppgave 3 (18% ~ 42 minutter) – Brukernavn passord innlogging

Vi skal se litt på bruken av passord i en webapp.

Anta at informasjon om brukere er lagret en tabell **bruker** i databasen (og tilsvarende **Bruker**-objekter i Java) med brukernavn (PK), passordhash, passordsalt, og diverse annen info.

Vi har tre ulike hjelpeklasser som du kan bruke i løsningene dine:

BrukerDAO henter og lagrer data i databasen, med metodene:

- `Bruker hentBruker(String brukernavn)` //Null om ikke finnes
- `void lagreNyBruker(Bruker nyBruker)`

PasswordUtil hjelper oss med kryptografi og sikkerhet, med metodene:

- `static void oppdaterBrukerMedPassord(Bruker b, String klartekstpass)`
Denne salter og hasher passordet, og lagrer det i brukerobjektet
- `static boolean erPassordGyldig(Bruker b, String klartekstpass)`
Denne sjekker om oppgitt passord er korrekt for denne brukeren

InnloggingUtil hjelper oss med innlogging, utlogging osv.:

- `static void loggInn(Bruker b, HttpServletRequest r)`

- a) Skriv det som mangler i `doPost()`-metoden som registrerer en ny bruker i applikasjonen. Etter registrering skal brukeren logges inn og gå til "hovedsiden".

```
@EJB BrukerDAO brukerDAO;

void doPost( ...) {
    Bruker bruker = ...           //Opprettet fra diverse validert input
    String passord = ...          //Hentet og validert fra brukerinput

    /* Skriv det som mangler */
}
```

- b) Skriv det som mangler i `doPost()`-metoden som sjekker om brukernavn og passord er gyldig ved forsøk på innlogging. Hvis bruker ikke finnes eller passordet ikke stemmer skal brukeren gå til "innlogging" igjen. Ellers skal brukeren logges inn og gå til "hovedsiden".

```
@EJB BrukerDAO brukerDAO;

void doPost( ...) {
    String brukernavn = ...       //Hentet og validert fra brukerinput
    String passord = ...          //Hentet og validert fra brukerinput

    /* Skriv det som mangler */
}
```

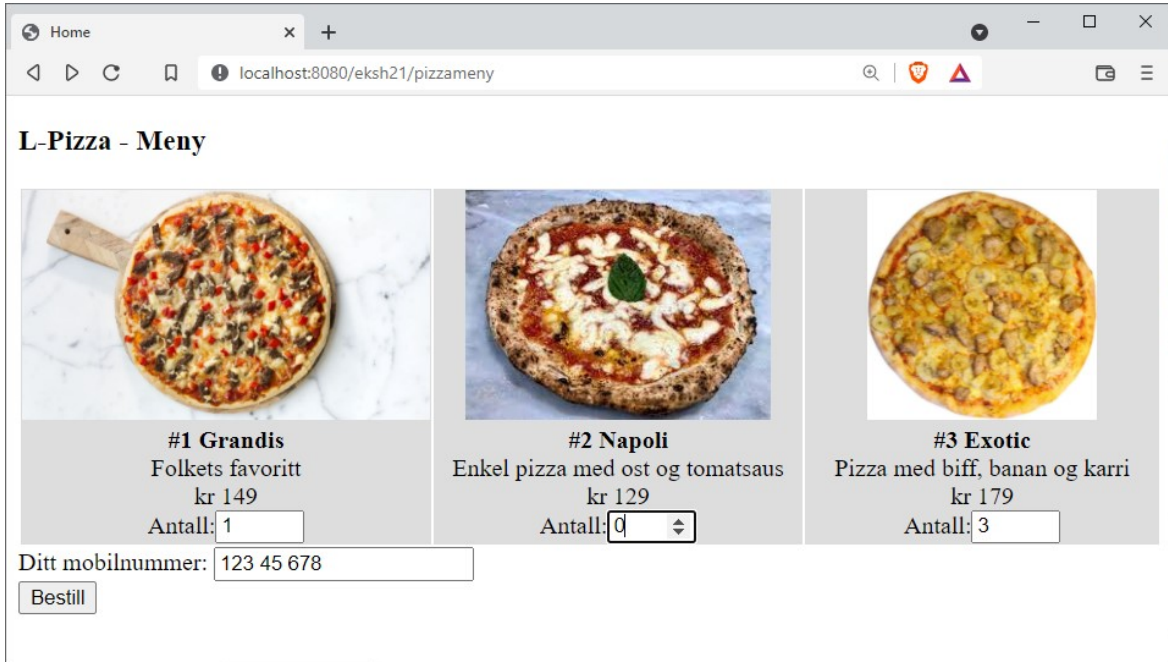
- c) Beskriv helt overordnet hva som gjennomføres i `PasswordUtil.erPassordGyldig()` for å sjekke om et passord er gyldig. Vis gjerne en løsning i Java. Du kan anta at `PasswordUtil` har private hjelpemetoder for hashing og salting som du kan bruke i løsningen din.

Oppgave 4 (24% ~ 58 minutter) – Webapplikasjoner, tjenerside

Dere skal lage litt av en løsning for håndtering av bestillinger hos en pizza-restaurant.

Merk: Utgangspunktet ligner på en tidligere eksamen, men les oppgaveteksten godt, og ikke gjør antagelser basert på den tidligere eksamenen.


Vi kan tenke oss at vi har en hovedside som på bildet under der kunder kan bestille pizzaer. Når kunden utfører bestillingen vil denne lagres i en database. PS! Det å bestille pizza er ikke en del av eksamen, men er tatt med for å gi kontekst til problemstillingen.




Home x +

localhost:8080/eksh21/pizzameny


L-Pizza - Meny



#1 Grandis
Folkets favoritt
kr 149
Antall:



#2 Napoli
Enkel pizza med ost og tomat saus
kr 129
Antall:



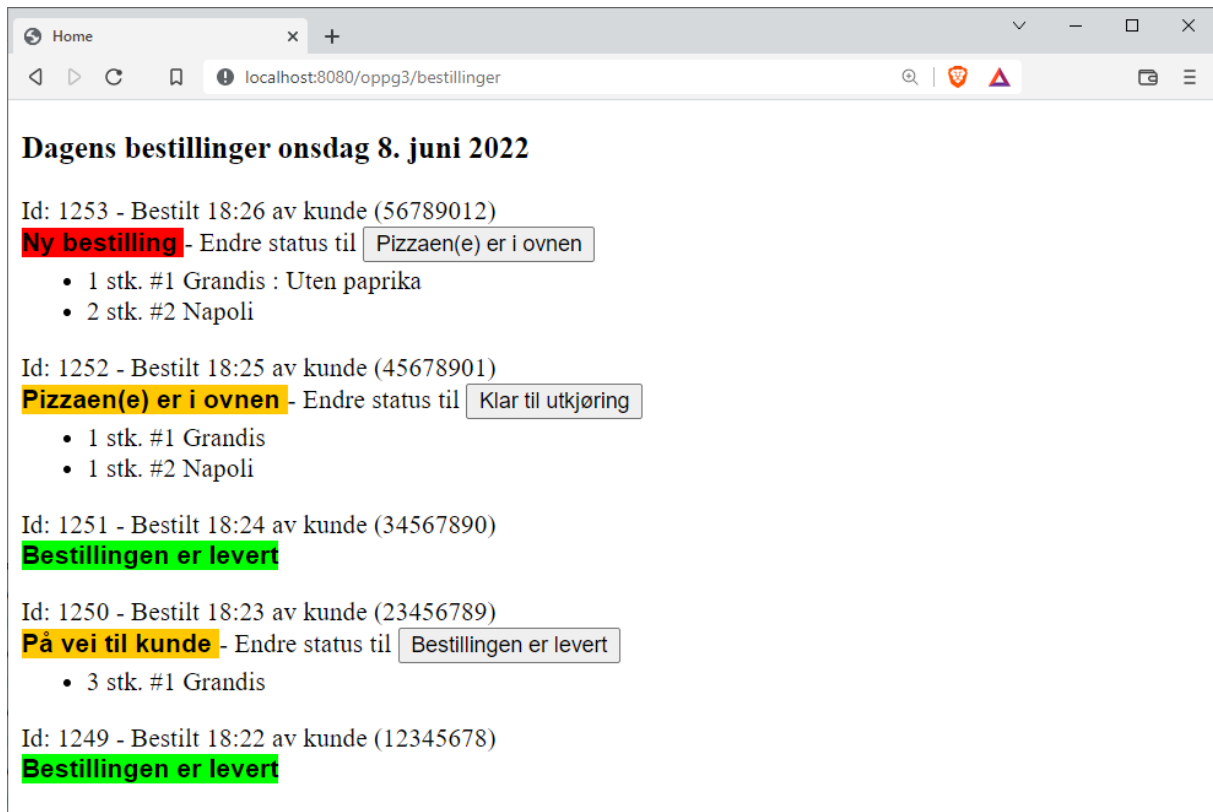
#3 Exotic
Pizza med biff, banan og karri
kr 179
Antall:

Ditt mobilnummer:

Bilde 4.1 - /pizzameny

I denne eksamensoppgaven skal vi lage en nettside for de som jobber på pizza-restauranten (pizza-bakere og -sjåførere), der de kan se og oppdatere status på bestillinger fra **Ny bestilling** via **Pizzaen(e) er i ovnen**, **Klar til utkjøring**, **På vei til kunde** og til slutt **Bestillingen er levert**.

Eksempel på en slik side kan være som bildet under:



Bilde 4.2 - /bestillinger

Vi har en hjelpeklasse (@Stateless EJB) **BestillingerDAO** som vi kan bruke til å hente bestillinger fra databasen og til å oppdatere status for en bestilling. Metodene som kan brukes ser slik ut:

- **public List<Bestilling> hentDagensBestillinger()** //Henter alle dagens bestillinger
- **public void oppdaterStatusForBestilling(int bestillingId)** //Oppdater til neste statusnivå

En oppdatert liste av bestillinger skal hentes fra databasen ved **hver** GET-request.

Et **Bestilling**-objekt inneholder følgende data

- | | |
|-----------------------------------|---|
| - int id | - Unik id for bestillingen |
| - Status status | - Bestillingens status (forklares mer nedenfor) |
| - LocalTime bestiltKlokken | - Bestillingens tidspunkt |
| - String kundemobil | - Kundens mobilnr |
| - List<String> pizzaer | - En forenklet tekst-representasjon av en ordrelinje, f.eks
"1 stk. #1 Grandis : Uten paprika" |

og konstruktører og metoder **etter behov**.

Status er implementert som en Enum, og har følgende innhold:

NY, LAGES, KLAR, PAA_VEI, LEVERT;

```
public String getTekst()      //Henter ut tekst for status, f.eks. "På vei til kunde" for
                              Status.PAA_VEI

public String getFarge()      //Henter ut hex-verdi for farge, f.eks. "#ff0000" (rød) for
                              Status.NY (RGB-koder brukt for økt fleksibilitet)

public Status getNextStatus() //Henter ut neste status, f.eks. Status.LAGES vil returneres
                              for Status.NY siden Lages kommer etter Ny.

public boolean isLevert()     //Om bestillingen er levert. Kun true for Status.LEVERT.
```

Eksempel på bruk i Java (hvis Enum er litt ukjent):

```
Status s = Status.NY;
System.out.println(s);           // NY
System.out.println(s.getTekst()); // Ny bestilling
System.out.println(s.getFarge()); // #ff0000
System.out.println(s.getNextStatus()); // LAGES
System.out.println(s.isLevert());  // false
```

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, ufarliggjøring** av brukerinput, **unntakshåndtering, god bruk av hjelpeklasser, elegant kode**, osv ...

Oppgaver

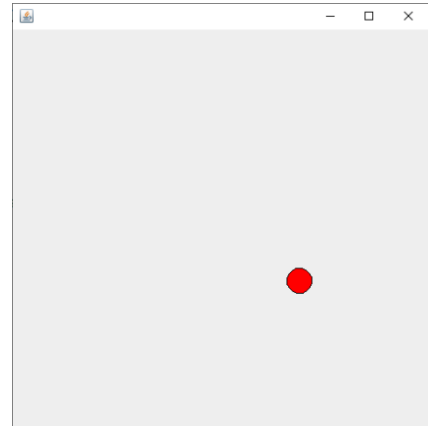
- (8%) Skriv **BestillingerServlet** som mappes til URLen **/bestillinger**. Denne skal ha ansvar for både det å få frem bestillingsoversikten og for å håndtere forespørsler om å oppdatere status for en bestilling. En oppdatert liste av bestillinger skal hentes fra databasen ved hver GET-request. Tilhørende view er `bestillinger.jsp`.
- (16%) Skriv **bestillinger.jsp** som gir en side tilsvarende Bilde 4.2.

Tips: For å få tekst uthevet med bakgrunnsfarge kan du få "jukse" ved å bruke denne koden:
Tekst med rød bakgrunn

Oppgave 5 (10% ~ 24 minutter) – Tråder

I denne oppgaven skal vi se på hvordan vi kan styre en ball med konsoll-kommandoer. I utgangspunktet har vi en **Ball**-klasse som viser en ball i bevegelse i et grafisk vindu:

```
public static void main(String[] args) {
    Ball ball = new Ball();
    ball.settOppGUI();
    ball.animér();
}
```



Slik det er nå vil ikke animér()-metoden returnere før vi lukker det grafiske vinduet.

Vi tenker oss å kjøre at animér() i en egen tråd slik at vi kan taste inn kommandoer for å "styre" ballen etter at denne er satt i gang. Vi tenker at Ball utvides med metodene **gass()** og **brems()** som endrer farten på ballen, og **avslutt()** som stopper animasjonen og lukker vinduet. Du kan anta at disse virker slik de skal.

Endre / utvid main() slik at vi kan "styre" farten på ballen med kommandoene **"g"** og **"b"**, og avslutte med kommandoen **"exit"**. Strukturen på main kan være ca. slik:

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

    /* Kode inn her */

    Scanner input = new Scanner(System.in);

    String kommando = input.nextLine();
    while (!kommando.equals("exit")) {

        /* Kode inn her */

        kommando = input.nextLine();
    }

    /* Kode inn her */
}
```

EKSAMENSOPPGÅVE

Emnekode: DAT108

Emnenamn: Programmering og webapplikasjoner

Klasse: 2. klasse DATA / INF

Dato: 8. juni 2022

Eksamensform: Skriftleg heimeeksamen (Wiseflow | FLOWassign)

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

Tal på eksamensoppgåver: 5

Tal på sider (medrekna denne): 10

Tal på vedlegg: Ingen

Tillatte hjelpemiddel: Alle.

Lærarar: 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" løysingar på oppgåver.

LUKKE TIL!

Oppg ve 1 (24% ~ 58 minutt) – Lambda-uttrykk og straumer

Generelt for alle sp rsm l i Oppg ve 1: For   f  god utteljing m  l ysingane ikkje vera un dig kompliserte, og det m  brukast lambdauttrykk, metodereferansar og streams der det passar.

a) Kva er datatypane til f lgjande uttrykk:

- i. `a -> a.length()`
- ii. `(Integer a) -> String.valueOf(a)`
- iii. `a -> a.getDayOfWeek().equals(DayOfWeek.MONDAY)`

Om mogleg, vis  g korleis uttrykka over kan omskrivast til metodereferansar.

b) Anta at vi har ei liste av **Person-objekt** referert av liste-variabelen **vener**, og vi  nsker   f  skrive dei ut sortert p  eit par ulike m tar. Person-klassen ser du nedanfor.

```
public class Person {  
    String fulltNamn;  
    LocalDate foddato;  
  
    // + konstrukt rar, gettarar, settarar og toString  
}
```

- i. Skriv ein main-metode som skriv lista ut sortert p  namn, ein person per linje.
- ii. Skriv ein main-metode som skriv lista ut sortert etter n r p   ret dei har bursdag ( ret betyr ikkje noko, berre n r p   ret) TIPS: Sidan  ret ikkje skal vera med i samanlikning kan dette setjast til eit fast  r i samanlikninga ved   bruka `LocalDate.withYear()`-metoden.

- c) Denne oppgåva går ut på å parse og summera saman alle positive heiltal i ein tekst.

Vi kan t.d. ha teksten "Per 6 32 Anne xyz Bergen 14" som skal gi svaret 52 (altså $6+32+14$).

Det første steget kan vera å dela opp teksten i tokens med `t.split(" ")`.

Deretter er det å plukka ut det som er tal. Til dette har eg definert eit predikat som seier om ein tekst er eit heiltal. Dette blir forventa brukt i løysinga.

```
Predicate<String> erEtPositivtHeltall = t -> t.matches("^\\d+$");
```

Lag ein `main()`-metode som tar utgangspunkt i ein tekst (lagra i ein String-variabel), og som ved hjelp av tipsa over bereknar summen av alle positive heiltal i teksten. Bruk streams-APIet. Svaret skal til slutt skrivast ut på skjermen.

- d) Vi har laga ein metode for å skriva ut ein tabell av funksjonsverdiar for kvadratfunksjonen $f(x) = x^2$:

```
public class Printer {
    public static void printTabellForKvadratfunksjon(
        double start, double stopp, double steg) {

        for (double x=start; x<=stopp; x+=steg) {
            double fx = x*x;
            System.out.printf("%8.3f%8.3f\n", x, fx);
        }
    }
}
```

Bruken av denne kan t.d. vera slik (med resultat til høgre):

```
public static void main(String... blablabla) {
    Printer.printTabellForKvadratfunksjon(1, 5, 1);
}
```

1,000	1,000
2,000	4,000
3,000	9,000
4,000	16,000
5,000	25,000

Vi ønsker å laga ein meir generell metode som skriv ut ein tabell av funksjonsverdiar for ein vilkårleg funksjon (t.d. $f(x) = \sin(x)$, $f(x) = \log(x)$, osv..).

- Lag ein metode tilsvarande den som er vist over slik at dette er mogleg. Kall den nye metoden `printTabellForFunksjon(...)`.
- Bruk den nye generelle metoden frå i. i main-programmet i staden for den gamle til å skriva ut kvadrattala frå 1 til 5 som i dømet.

Oppgave 2 (24% ~ 58 minutt) – JavaScript

- a) Ein applikasjon for ei stoppeklokke inneheld JavaScript-koden under:

```
class TimerController {  
    #start = Date.now();  
  
    constructor(container) {  
        const bt = container.querySelector("button");  
        bt.addEventListener("click", this.runde);  
    }  
  
    runde() {  
        console.log(Date.now() - this.#start);  
    }  
}
```

Applikasjonen har ein knapp for å vise rundetider i web-konsollen.

Du kan anta at den tilhøyrande HTML-koden er riktig laga og inkluderer JavaScript-koden på riktig måte. Det er heller ingen syntaksfeil i JavaScript-koden.

- Konstruktøren inneheld ein feil som gjer at applikasjonen ikkje kan verka. Gjer greie for kva problemet er.
 - Omskriv koden og fjern feilen ved å bruke **Function** sin metode *bind*.
 - Problemet kan også løysast ved å kalle *runde* i ein kringsluttande funksjon. Gjer greie for kvifor eit funksjonsuttrykk med pil-notasjon kan løyse problemet, mens eit standard funksjonsuttrykk kan gje problem.
- b) Opprett ei JavaScript-klasse **Ordanalyse**. Instansar av klassa skal kunne analysere ein tekst, der du berre skal implementere tre metodar, *setTekst*, *getAntallord* og *getOrdliste*.

Metoden *setTekst* verte nytta for å registrere teksten som skal verte analysert, metoden *getAntallord* skal returnere tal på unike ord i teksten, og metoden *getOrdliste* skal returnere ei liste av alle unike ord i teksten.

Ordanalyse skal ikkje skilje mellom store og små bokstaver når unike ord verte registrerte eller talde, og lista med ord som verte returnert av *getOrdliste* skal berre nytte små bokstavar.

Døme med kode nedanfor viser bruk av klassa **Ordanalyse**:

```
const oa = new Ordanalyse();  
oa.setTekst("Velkommen! Nå, ja; ja nå er det eksamenstid.");  
const antall = oa.getAntallord();  
const tabell = oa.getOrdliste();
```

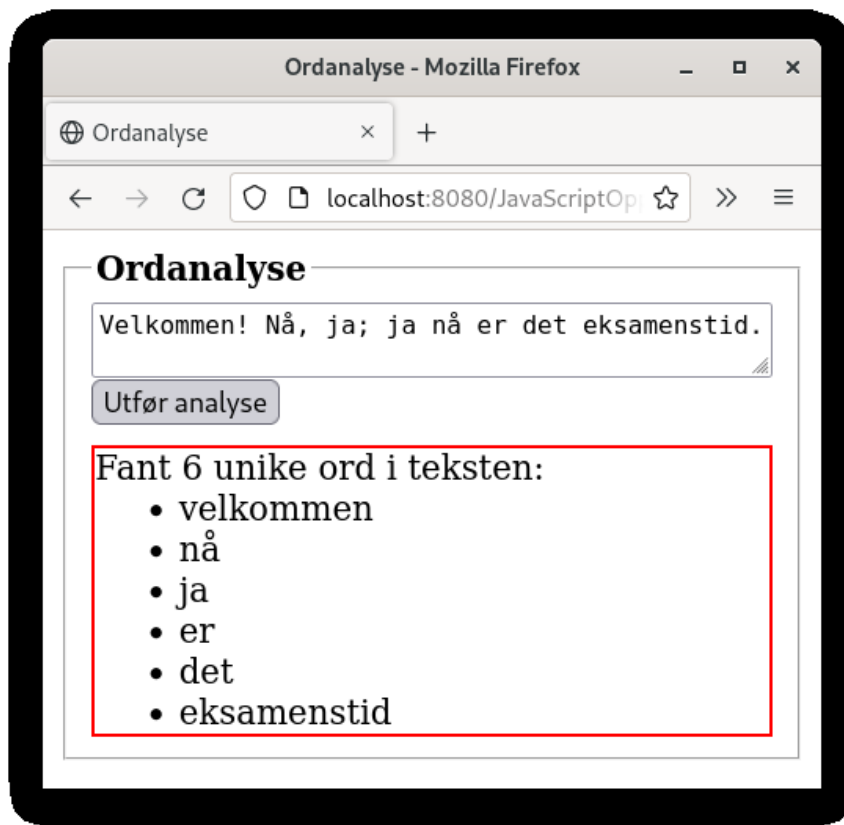
Variabelen *antall* vil nå ha verde 6, og variabelen *tabell* skal ha følgjande innhald:

```
["velkommen", "nå", "ja", "er", "det", "eksamenstid"]
```

Ein løysing for å registrere unike ord er å nytte ordet som nøkkel (key) i ei JavaScript **Map**.

Oppgåve: Skriv JavaScript-kode for **Ordanalyse** med metodane *setTekst*, *getAntallord* og *getOrdliste* i samsvar med teksten over.

- c) Figuren nedenfor viser ei webside som bruker JavaScript-klassa **Ordanalyse** frå føregående oppgåve:



Figur 2: Webside for ordanalyse

Form-elementet som verte vist i figuren over er gitt av følgjande HTML-kode:

```
<form>
  <fieldset>
    <legend>Ordanalyse</legend>
    <textarea placeholder="Fyll inn tekst" data-tekst></textarea>
    <button type="button" data-analyser>Utfør analyse</button>
    <div data-resultat></div>
  </fieldset>
</form>
```

Ved klikk på knappen «Utfør analyse», eller når websida verte lasta på nytt skal JavaScript-kode produsere innhaldet som er vist inne i den røde ramma i figuren.

Oppgåve: Skriv JavaScript-kode for å implementer funksjonaliteten som er omtala over. Løysinga må nytte JavaScript-klassa **Ordanalyse** frå føregående oppgåve.

Oppg ve 3 (18% ~ 42 minutt) – Brukernamn passord innlogging

Vi skal sj  litt p  bruken av passord i ein webapp.

Anta at informasjon om brukarar er lagra ein tabell **bruker** i databasen (og tilsvarande **Bruker**-objekt i Java) med brukarnamn (PK), passordhash, passordsalt, og diverse annan info.

Vi har tre ulike hjelpeklassar som du kan bruka i l ysingane dine:

BrukerDAO hentar og lagrar data i databasen, med metodane:

- `Bruker hentBruker(String brukernavn)` //Null om ikkje finnst
- `void lagreNyBruker(Bruker nyBruker)`

PassordUtil hjelpar oss med kryptografi og trygging, med metodane:

- `static void oppdaterBrukerMedPassord(Bruker b, String klartekstpass)`
Denne saltar og hasher passordet, og lagrar det i bruker-objektet
- `static boolean erPassordGyldig(Bruker b, String klartekstpass)`
Denne sjekker om oppgitt passord er korrekt for denne brukaren

InnloggingUtil hjelper oss med innlogging, utlogging osv.:

- `static void loggInn(Bruker b, HttpServletRequest r)`

- a) Skriv det som manglar i `doPost()`-metoden som registrerer ein ny brukar i applikasjonen. Etter registrering skal brukaren loggast inn og g  til "hovudsida".

```
@EJB BrukerDAO brukerDAO;

void doPost( ...) {
    Bruker bruker = ...           //Oppretta fr  diverse validert input
    String passord = ...          //Henta og validert fr  brukarinput

    /* Skriv det som manglar */
}
```

- b) Skriv det som manglar i `doPost()`-metoden som sjekkar om brukarnamn og passord er gyldig ved fors k p  innlogging. Viss brukar ikkje finst eller passordet ikkje stemmer skal brukaren g  til "innlogging" igjen. Elles skal brukaren loggast inn og g  til "hovudsida".

```
@EJB BrukerDAO brukerDAO;

void doPost( ...) {
    String brukernavn = ...        //Henta og validert fr  brukarinput
    String passord = ...           //Henta og validert fr  brukarinput

    /* Skriv det som manglar */
}
```

- c) Beskriv heilt overordna kva som blir gjennomf rt i `PassordUtil.erPassordGyldig()` for   sjekka om eit passord er gyldig. Vis gjerne ei l ysing i Java. Du kan anta at `PassordUtil` har private hjelpemetodar for hashing og salting som du kan bruka i l ysinga di.

Oppg ve 4 (24% ~ 58 minutt) – Webapplikasjoner, tenerside

De skal laga litt av ei l ysing for handtering av bestillinger hos ein pizza-restaurant.


Merk: Utgangspunktet liknar p  ein tidlegare eksamen, men les oppg veteksten godt, og ikkje gjer antagelser basert p  den tidlegare eksamenen.

Vi kan tenka oss at vi har ei hovudsider som p  bildet under der kundar kan bestilla pizzaer. N r kunden utf rer bestillingen vil denne bli lagra i ein database. PS! Det   bestilla pizza er ikkje ein del av eksamen, men er tatt med for   gi kontekst til problemstillinga.


Home x +

localhost:8080/eksh21/pizzameny


L-Pizza - Meny



#1 Grandis
Folkets favoritt
kr 149
Antall:



#2 Napoli
Enkel pizza med ost og tomat saus
kr 129
Antall:



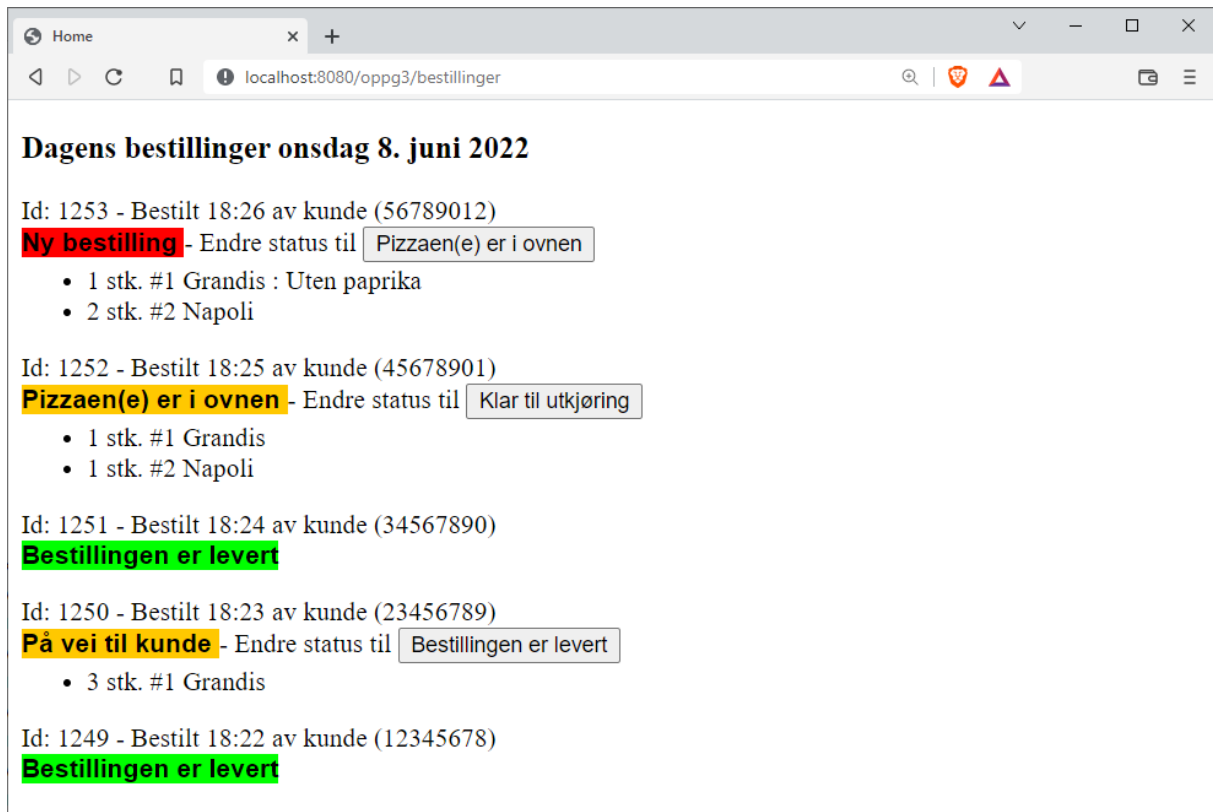
#3 Exotic
Pizza med biff, banan og karri
kr 179
Antall:

Ditt mobilnummer:

Bilde 4.1 - /pizzameny

I denne eksamensoppgåva skal vi laga ei nettside for dei som jobbar på pizza-restauranten (pizza-bakarar og -sjåførar), der dei kan sjå og oppdatere status på bestillinger frå **Ny bestilling** via **Pizzaen(e) er i ovnen**, **Klar til utkjøring**, **På vei til kunde** og til slutt **Bestillingen er levert**.

Døme på ei slik side kan vera som bildet under:



Bilde 4.2 - /bestillinger

Vi har ein hjelpeklasse (@Stateless EJB) **BestillingerDAO** som vi kan bruka til å henta bestillinger frå databasen og til å oppdatere status for ein bestilling. Metodane som kan brukast ser slik ut:

- **public List<Bestilling> hentDagensBestillinger()** //Henter alle dagens bestillinger
- **public void oppdaterStatusForBestilling(int bestillingId)** //Oppdater til neste statusnivå

Ei oppdatert liste av bestillinger skal hentast frå databasen ved **kvar** GET-request.

Et **Bestilling**-objekt inneheld følgjande data

- | | |
|-----------------------------------|---|
| - int id | - Unik id for bestillingen |
| - Status status | - Bestillingens status (blir forklart meir nedanfor) |
| - LocalTime bestiltKlokken | - Bestillingens tidspunkt |
| - String kundemobil | - Kundens mobilnr |
| - List<String> pizzaer | - Ein forenklet tekst-representasjon av ei ordrelinje, t.d.
"1 stk. #1 Grandis : Uten paprika" |

og konstruktørar og metodar **etter behov**.

Status er implementert som ein Enum, og har følgjande innhald:

NY, LAGES, KLAR, PAA_VEI, LEVERT;

```
public String getTekst()      //Henter ut tekst for status, t.d. "På vei til kunde" for
                              Status.PAA_VEI

public String getFarge()     //Henter ut hex-verdi for farge, t.d. "#ff0000" (raud) for
                              Status.NY (RGB-koder brukt for auka fleksibilitet)

public Status getNextStatus() //Henter ut neste status, t.d. Status.LAGES vil returnerast
                              for Status.NY sidan Lages kjem etter Ny.

public boolean isLevert()    //Om bestillingen er levert. Berre true for Status.LEVERT.
```

Døme på bruk i Java (viss Enum er litt ukjent):

```
Status s = Status.NY;
System.out.println(s);           // NY
System.out.println(s.getTekst()); // Ny bestilling
System.out.println(s.getFarge()); // #ff0000
System.out.println(s.getNextStatus()); // LAGES
System.out.println(s.isLevert());  // false
```

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, **trådsikkerhet, robusthet, ufarliggjøring** av brukarinput, **unntakshandtering, god bruk av hjelpeklasser, elegant kode**, osv ...

Oppgåver

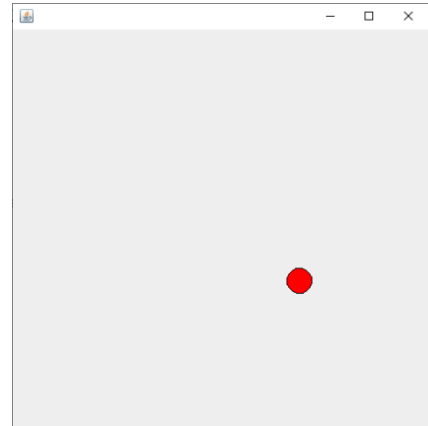
- (8%) Skriv **BestillingerServlet** som mappes til URLen **/bestillinger**. Denne skal ha ansvar for både det å få fram bestillingsoversikten og for å handtera førespurnader om å oppdatere status for ein bestilling. Ei oppdatert liste av bestillinger skal hentast frå databasen ved kvar GET-request. Tilhøyrande view er bestillinger.jsp.
- (16%) Skriv **bestillinger.jsp** som gir en side tilsvarande Bilde 4.2.

Tips: For å få tekst utheva med bakgrunnsfarge kan du få "juksa" ved å bruka denne koden:
Tekst med rød bakgrunn

Oppgave 5 (10% ~ 24 minutt) – Trådar

I denne oppgåva skal vi sjå på korleis vi kan styra ein ball med konsoll-kommandoar. I utgangspunktet har vi ein **Ball**-klasse som viser ein ball i bevegelse i eit grafisk vindauge:

```
public static void main(String[] args) {  
    Ball ball = new Ball();  
    ball.settOppGUI();  
    ball.animér();  
}
```



Slik det er no vil ikkje animér()-metoden returnera før vi lukkar det grafiske vindauget.

Vi tenker oss å køyra animér() i ein eigen tråd slik at vi kan tasta inn kommandoar for å "styra" ballen etter at denne er satt i gang. Vi tenker at Ball blir utvida med metodane **gass()** og **brems()** som endrar farten på ballen, og **avslutt()** som stoppar animasjonen og lukkar vindauget. Du kan anta at desse verkar slik dei skal.

Endre / utvid main() slik at vi kan "styra" farten på ballen med kommandoane **"g"** og **"b"**, og avslutta med kommandoen **"exit"**. Strukturen på main kan vera ca. slik:

```
public static void main(String[] args) {  
  
    /* Kode inn her */  
  
    Scanner input = new Scanner(System.in);  
  
    String kommando = input.nextLine();  
    while (!kommando.equals("exit")) {  
  
        /* Kode inn her */  
  
        kommando = input.nextLine();  
    }  
  
    /* Kode inn her */  
}
```