

Løsningsforslag

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.

Løsning:

i.

Den "eneste" Java-klassen som har metoden `length()` er `String`.

Så da blir det en funksjon fra `String` til `Integer`:

```
2    Function<String, Integer> lengde1 = a -> a.length();
```

Som metoderreferanse:

```
2    Function<String, Integer> lengde2 = String::length;
```

ii.

Siden `String.valueOf()` kommer i mange overlastede utgaver ble det her oppgitt hvilken datatype `a` er. Vi får da en funksjon fra `Integer` til `String`:

```
2    Function<Integer, String> somTekst1 = (Integer a) -> String.valueOf(a);
```

Som metoderreferanse (som virker for alle lovlige typer av `a`):

```
2    Function<Integer, String> somTekst2 = String::valueOf;
```

iii.

Her må jo `a` være et objekt som har metoden `getDayOfWeek()`. Aktuelle kandidater er `LocalDateTime` og `LocalDate`. Så f.eks.:

```
2    Predicate<LocalDate> erMandag  
    = a -> a.getDayOfWeek().equals(DayOfWeek.MONDAY);
```

Merknad:

Ganske mange besvarelser sier at datatypene til uttrykkene er ting som `int`, `String` og `boolean`. Dette er feil. Dette er datatypene til returverdiene man får ved å anvende uttrykkene på en inputverdi.

Uttrykkene i seg selv representerer objekter med én metode (altså en anonym funksjon) som kan anvendes på ulike data.

Det kan dessverre ikke gis poeng for svar som `int`, `String` og `boolean`.

- 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, getter, 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.

Løsning:

```
public static void main(String[] args) {  
    List<Person> venner = ...  
  
    i.  
    2 Comparator<Person> paaNavn =  
        (p1, p2) -> p1.getFulltNavn().compareTo(p2.getFulltNavn());  
  
    Alternativt:  
    Comparator<Person> paaNavn =  
        Comparator.comparing(Person::getFulltNavn);  
  
    3 venner.stream().sorted(paaNavn).forEach(System.out::println);  
  
    ii.  
    3 Comparator<Person> paaBursdag =  
        (p1, p2) -> p1.getFoddato().withYear(2000)  
            .compareTo(p2.getFoddato().withYear(2000));  
  
    2 venner.stream().sorted(paaBursdag).forEach(System.out::println);  
}
```

Merknad:

Bruker man `Collections.sort(liste)` eller `liste.sort()` i stedet for `liste.stream().sorted()` gis litt trekk siden man ikke er bedt om å sortere listen, og siden oppgaven sier *"For å få god uttelling må ... det må brukes streams der det passer."*. Her passer det godt med streams.

En del besvarelser bruker `map()`, f.eks. slik:

```
    venner.stream().map(Person::getFulltNavn).sorted(). ...
```

Dette blir feil siden `map()` her omformer en strøm av personer til en strøm av stringer. Man kan ikke få ut en liste av personer i andre enden når man gjør slik.

- 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 deres.

```
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`t. Svaret skal til slutt skrives ut på skjermen.

Løsning:

```
public static void main(String... blablabla) {  
    String tekst = "Per 6 32 Anne xyz Bergen 14";  
    Predicate<String> erEtPositivtHeltall = t -> t.matches("^\\d+$");  
3    int sum = Arrays.stream(tekst.split(" "))  
2        .filter(erEtPositivtHeltall)  
2        .mapToInt(Integer::parseInt) // evt. Integer::valueOf  
2        .sum();  
  
    Alternativt uten mapToInt:  
        .map(Integer::parseInt)  
        .reduce(0, Integer::sum);  
1    System.out.println(sum);  
}
```

Merknad:

Løkker og if-setninger osv.. er ikke poenggivende her. Oppgaven handler om funksjonell programmering med anonyme funksjoner og streams,

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.

Løsning:

```
i.
3 public static void printTabellForFunksjon(
    Function<Double, Double> funksjon,
    double start, double stopp, double steg) {

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

Alternativt:

```
public static void printTabellForFunksjon2( ... ) {
    Stream.iterate(start, x -> x+steg).takeWhile(x -> x<=stopp)
        .forEach(x -> System.out.printf("%8.3f%8.3f\n", x, funksjon.apply(x)));
}
```

```
ii.
4 Printer.printTabellForFunksjon(x -> x*x, 1, 5, 1);
```

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.

- i. Konstruktøren inneholder en feil som gjør at applikasjonen ikke kan virke. Utdyp hva problemet er.
- ii. Omskriv koden og fjern feilen ved å bruke **Function** sin metode *bind*.
- iii. 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.

Løsning:

i.

Kall av metoden *this.runde* er et eksempel på asynkron kode. Metoden skal bli utført senere når knappen referert ved **bt** blir klikket på. Da er *this* en referanse til knappen **bt**, ikke instansen av **TimerController**. Knappen **bt** har ingen metode *runde*, og koden vil feile.

ii.

```
constructor(container) {  
  const bt = container.querySelector("button");  
  bt.addEventListener("click", this.runde.bind(this))  
}
```

iii.

Koden under viser en løsning med en omsluttende funksjon med pil-notasjon:

```
constructor(container) {  
  container.querySelector("button")  
    .addEventListener("click", () => {this.runde()})  
}
```

Funksjon ved pil-notasjon vil ikke binde *this* og verdien er gitt av den omsluttende konteksten, og vil altså være en instans av **TimerController**. Et standard funksjonsuttrykk vil derimot gi ny verdi til *this*, gitt av funksjonen sin kontekst som her er knappen som ble klikket.

- 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.

Løsning:

```
class Ordanalyse {
  #ordset = new Set();

  setTekst(tekst) {
    this.#ordset.clear();
    if (tekst === "") return;

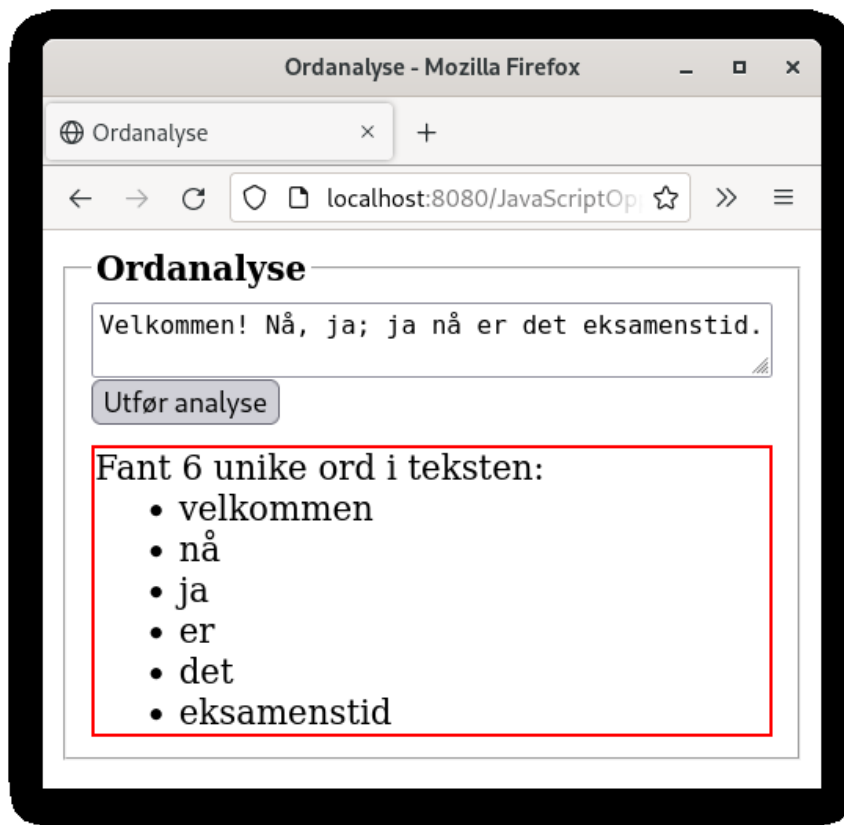
    const tabell = tekst.split(/\P{Letter}+/u)
      .filter(ord => /\p{Letter}/u.test(ord))
      .map(ord => ord.toLocaleLowerCase());

    tabell.forEach((ord) => { this.#ordset.add(ord) });
  }

  getAntallord() {
    return this.#ordset.size;
  }

  getOrdliste() {
    return Array.from(this.#ordset.values());
  }
}
```


- 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.

Løsning:

```
class Controller {
  #tekstElement;
  #btAnalyser;
  #ordanalyse;
  #resultatElement;

  constructor(rootElement) {
    this.#tekstElement = rootElement.querySelector("[data-tekst]");
    this.#btAnalyser = rootElement.querySelector("[data-analyser]");
    this.#resultatElement = rootElement.querySelector("[data-resultat]");

    this.#btAnalyser
      .addEventListener("click", this.#analyser.bind(this));
    this.#ordanalyse = new Ordanalyse();

    this.#analyser();
  }

  #analyser() {
    this.#resultatElement.textContent = "";
    this.#ordanalyse.setTekst(this.#tekstElement.value);

    const antallOrd = this.#ordanalyse.getAntallord();
    if (antallOrd === 0) {
      this.#resultatElement.insertAdjacentHTML("beforeend",
        "<p>Finner ingen ord i tekstfeltet.</p>");
    } else {
      this.#resultatElement.insertAdjacentHTML("beforeend",
        "<p>Fant ${antallOrd} unike ord i teksten:</p>");
      this.#visListe();
    }
  }

  #visListe() {
    const ulElm = document.createElement("ul");
    this.#resultatElement.appendChild(ulElm);

    const ordliste = this.#ordanalyse.getOrdliste();
    ordliste.forEach(ord => {
      const liElm = document.createElement("li");
      liElm.textContent = ord;
      ulElm.appendChild(liElm);
    });
  }
}

function init() {
  new Controller(document.querySelector("fieldset"));
}

document.addEventListener("DOMContentLoaded", init);
```

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 i 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 passord med hashet passord i brukerobjektet

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 fra brukerinput

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

Løsning, se neste side:

...

```

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

1    /* Kunne evt. startet med en test på om ønsket brukernavn allerede
    * var registrert. Kan da f.eks. anta at vi har brukernavnet i en
    * String-variabel brukenavn hentet fra request-parameter og gjøre
    * noe slikt som:
    if (brukerDAO.hentBruker(brukenavn) != null) {
        response.sendRedirect("registreringssiden med feilmelding");
        return;
    }

    //Ved OK input må denne må gjøres først! for å få med alle data
2    PassordUtil.oppdaterBrukerMedPassord(bruker, passord);

    //Så kan vi lagre den nye brukeren i databasen
2    brukerDAO.lagreNyBruker(bruker);

    //Til slutt er det innlogging og omdirigering til hovedsiden
2    InnloginUtil.loggInn(bruker, request);
3    response.sendRedirect("hovedsiden");
}

```

- 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 fra brukerinput
    String passord = ...         //Hentet fra brukerinput

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

Løsning:

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

2   Bruker bruker = brukerDAO.hentBruker(brukernavn);
1+2  if (bruker == null || !PassordUtil.erPassordGyldig(bruker, passord)) {
2    response.sendRedirect("innlogginssiden med feilmelding");
    } else {
2    InnlogginUtil.loggInn(bruker, request);
1    response.sendRedirect("hovedsiden");
    }
}
```

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

Løsning:

For å sjekke om et oppgitt passord er gyldig for en bruker må man salte og hashe det oppgitte klartekstpassordet **med samme salt som er lagret for brukeren** og se om beregnet hash er lik den lagrede hashen for denne brukeren.

F.eks. slik:

```
boolean erPassordGyldig(Bruker bruker, String klartekstpassord) {
    return hashMedSalt(klartekstpassord, bruker.salt).equals(bruker.hash);
}
```

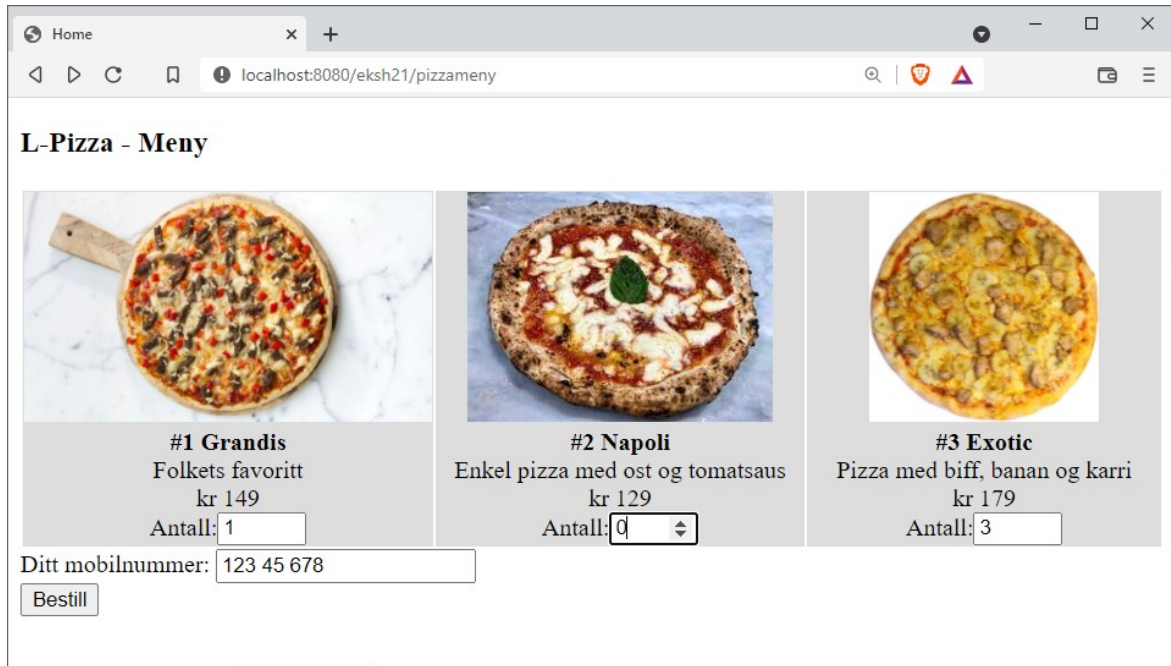
der `hashMedSalt()` er en hjelpemetode.

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:



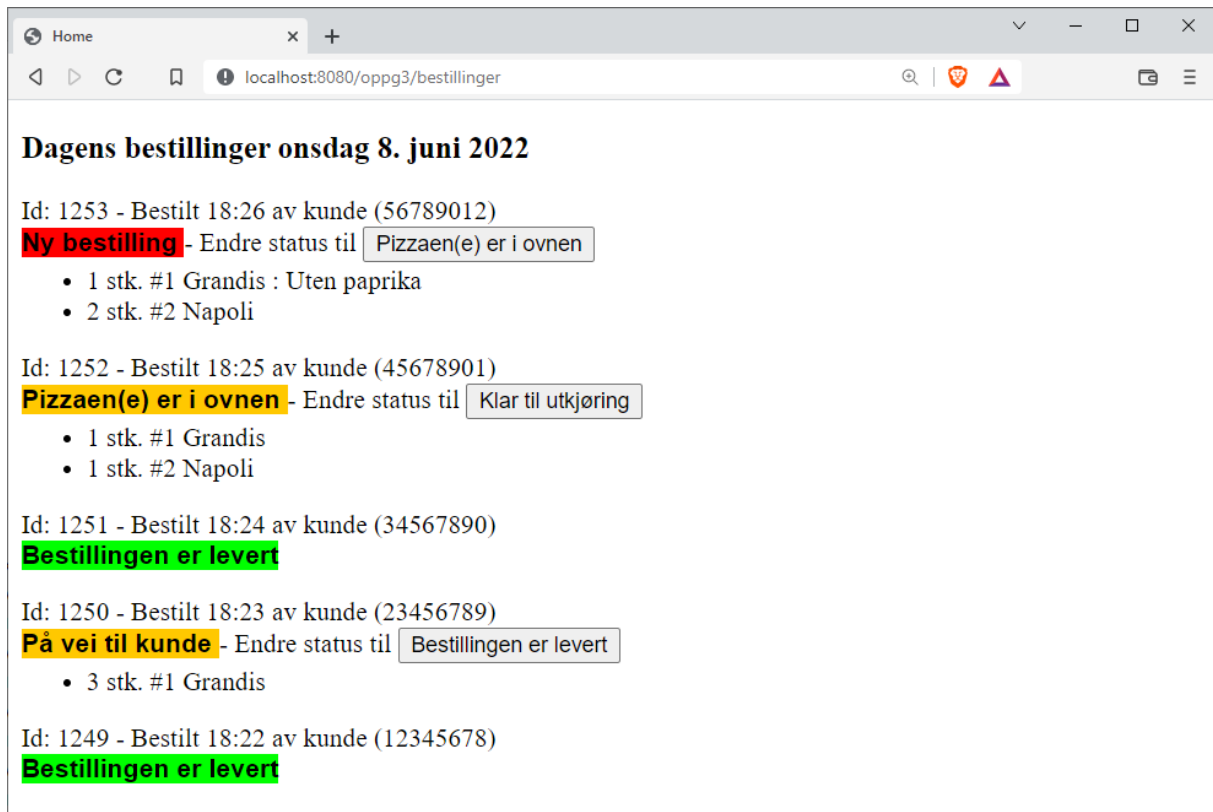
#3 Exotic
Pizza med biff, banan og karri
kr 179
Antall:

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 liste med forenklede tekst-representasjoner av ordrelinjene, 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

- a) (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.

Løsning:

```
1 @WebServlet("/bestillinger")
public class BestillingerServlet extends HttpServlet {

1  @EJB
    BestillingDAO dao;

    /** Controller for å hente frem bestillinger-siden */
    protected void doGet(... request, ... response) throws ... {

1        List<Bestilling> bestillinger = dao.hentDagensBestillinger();
2        request.setAttribute("bestillinger", bestillinger);

        LocalDate nowDato = LocalDate.now();
        String dagensDato = .. //en pent formatert utgave av nowDato
        request.setAttribute("dagensDato", dagensDato);

1        request.getRequestDispatcher("WEB-INF/jsp/bestillinger.jsp")
            .forward(request, response);
    }

    /** Controller for å oppdatere status for en bestilling
     * Aktuell bestillings-id er sendt med requesten som parameter. */
    protected void doPost(... request, ... response) throws ... {

1        int bestillingsid = Integer.parseInt(request.getParameter("bid"));
2        dao.oppdaterStatusForBestilling(bestillingsid);
1        response.sendRedirect("bestillinger");
    }
}
```

Merknader:

En del besvarelser henter inn bestillingsoversikten en gang for alle i en `init()`-metode, og lagrer den deretter i `servlet-context`. Dette fører jo til at ingenting blir oppdatert på siden (husk at listen er "detached" fra JPA sin `persistence-context`)! Det står tydelig i oppgaveteksten at *"En oppdatert liste av bestillinger skal hentes fra databasen ved hver GET-request"*.

Post-metoden skal (underforstått) i følge oppgaveteksten ha ansvar for *"å håndtere forespørsler om å oppdatere status for en bestilling"*. Skjermbildet (bilde 4.2) viser at hver bestilling har en knapp som kan trykkes for å sende en slik forespørsel. Den naturlige parameteren her er derfor `bestillingsid`!

b) (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:

```
<span bakgrunnsfarge="#ff0000">Tekst med rød bakgrunn</span>
```

Løsning:

```
<html>
<body>

    <h3>Dagens bestillinger ${dagensDato}</h3>

1   <c:forEach var="b" items="${bestillinger}">
1       <form method="post">
2           <input type="hidden" name="bid" value="${b.id}">

1           <p>Id: ${b.id} -
            Bestilt ${b.bestiltKlokken} av kunde (${b.kundemobil})<br>
1           <span bakgrunnsfarge="${b.status.farge}">${b.status.tekst}</span>

1           <c:if test="${not b.status.levert}">
            - Endre status til
1           <input type="submit" value="${b.status.nesteStatus.tekst}">

            <!-- Alternativt. Erstatte også hidden-parameter hvis gjort riktig -->
            <button name="bid" value="${b.id}">${b.status.nesteStatus.tekst}</button>

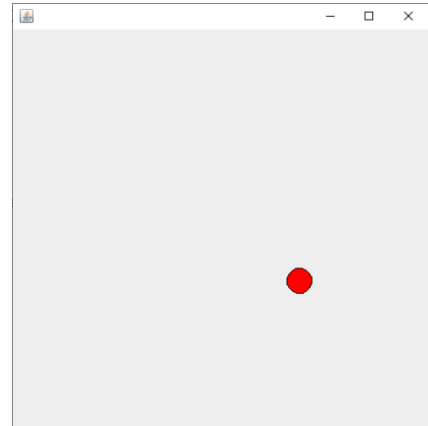
1           <ul><c:forEach var="pizza" items="${b.pizzaer}">
1               <li>${pizza}</li>
            </c:forEach></ul>
            </c:if>
        </form>
    </c:forEach>

</body>
</html>
```

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 */
}
```

Løsning:

```
public static void main(String[] args) throws ... {  
1    Ball ball = new Ball();  
    ball.settOppGUI();  
2    Runnable ballanimasjon = () -> ball.animér();  
2    Thread t = new Thread(ballanimasjon);  
2    t.start();  
  
    Scanner input = new Scanner(System.in);  
  
    String kommando = input.nextLine();  
    while (!kommando.equals("exit")) {  
        if (kommando.equals("g")) {  
1            ball.giGass();  
        } else if (kommando.equals("b")) {  
1            ball.brems();  
        }  
  
        kommando = input.nextLine();  
    }  
1    ball.avslutt();  
    t.join();  
}
```