

EKSAMENSOPPGAVE

Emnekode: DAT108

Emnenavn: Programmering og webapplikasjoner

Klasse: 2. klasse DATA / INF

Dato: 9. desember 2021

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 (20% ~ 48 minutter) – Lambda-uttrykk og strømmer

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

a) Tenk at du har en liste av hundenavn slik:

```
List<String> hunder = List.of("Fido", "Buster", "Colin");
```

Ved å bruke **forEach(c)**-metoden på listen med ulike parametere **c** kan du lage 3 ulike utskrifter, slik:

Fido Buster Colin	*Fido* *Buster* *Colin*	odiF retsuB niloC
-------------------------	-------------------------------	-------------------------

Skriv de 3 ulike variantene av parameteren **c** og lagre dem i variablene **printPlain**, **printMedStjerner** og **printBaklengs**. Husk å få med datatypen til variablene. *Tips til baklengs: new StringBuilder(s).reverse() vil snu strengen s.*

I de neste deloppgavene skal vi se på en liste av bøker. En Bok har disse egenskapene:

```
class Bok {
    public String tittel;
    public int aar;
    public List<String> forfattere; //Hver forfatter er en enkel String
    ... konstruktører og metoder ...
}
```

Tenk så at vi f.eks. har en liste med bøker slik:

```
List<Bok> boker = List.of(
    new Bok("Core Java Volume I", 2022, List.of("Cay Horstmann")),
    new Bok("Effective Java", 2017, List.of("Cay Horstmann")),
    ...
    new Bok("Head First Java", 2005,
        List.of("Kathy Sierra", "Bert Bates")),
    new Bok("Java Concurrency in Practice", 2006,
        List.of("Brian Goetz", "Tim Peierls", "Joshua Bloch"))
);
```

Du skal nå løse noen oppgaver ved å bruke streams og lambda-uttrykk.

- b) Skriv en setning som lager en ny liste av alle bøker som er utgitt i 2015 eller tidligere
- c) Skriv en setning som lager en liste av alle boktitler som inneholder ordet Java
- d) Skriv en setning som lager en liste av alle forfatterne uten duplikater
- e) Utvid d) med at forfatterne sorteres på etternavn ved å bruke `Stream<T> sin sorted(Comparator<? super T> comparator)` der comparatoren er tilordnet en variabel kalt **paaEtternavn** før bruk. Tips: `s.substring(s.lastIndexOf(" "))` gir etternavnet for navnet s.

Oppgave 2 (20% ~ 48 minutter) – JavaScript

- a) HTML taggen *script* kan brukes sammen med et attributt *defer*.
- Hva er konsekvensen av *defer* attributtet. Kan *defer* ha noe å si for ytelsen til webapplikasjonen? Svaret må begrunnes.
 - Hvor i et webdokument bør *script* tagg med *defer* plasseres? Svaret må begrunnes.
 - Hvor i et webdokument bør *script* tagg uten *defer* plasseres? Svaret må begrunnes.
- b) For å modifisere DOM-strukturen til et webdokument har vi blant annet egenskapene *innerHTML*, *textContent* og *innerText*, og metoden *insertAdjacentHTML*.
- Gi eksempler på når det er riktig å bruke *innerHTML* og *insertAdjacentHTML*. Svaret må begrunnes.
 - Gi eksempler på situasjoner der *innerHTML* og *insertAdjacentHTML* ikke må brukes. Svaret må begrunnes.
 - Gi eksempler på når det er riktig å bruke *textContent* og *innerText*. Svaret må begrunnes.
- c) Opprett en JavaScript klasse **Parking**. Instanser av klassen skal kunne administrere et parkeringsområde og parkeringsavgift for biler på området.

Et parkeringsområde har et gitt antall plasser for biler, og hver bil må tilhøre en takstgruppe. Klassen har metodene *addCar* og *removeCar* for å registrere at biler kjører inn og ut av parkeringsområdet.

Kodeeksempelet under demonstrer hvordan en instans av **Parking** kan bli opprettet:

```
const carpark= new Parking(50, { electric: 5, normal: 30 });
```

Første argument til konstruktøren er antall parkeringsplasser på parkeringsområdet, og andre argument er et objekt med takstgrupper og parkeringsavgift per påbegynt 15 minutt intervall. I eksempelet koster det 5 kroner per påbegynt 15 minutt intervall for biler i takstgruppe *electric* og 30 for biler i takstgruppe *normal*.

For biler som forlater området innen 10 minutter beregnes det ingen parkeringsavgift.

Klassen **Parking** har følgende metoder:

- addCar(regno,taxgroup)*: Metoden registrerer at bil med registreringsnummer *regno* som er i takstgruppe *taxgroup* har kjørt inn på parkeringsområdet.

Parametere:

- Inn-parametere: **String**, **String**
- Returverdi: **Object**

Metoden returnerer et objekt, eller *null*. Returverdi *null* betyr at registrering av bil feilet. Grunner for returverdi *null* kan være at parkeringsområdet er fullt, eller at takstgruppen *taxgroup* ikke finnes.

Kodeeksempelet under registrerer at bilen «EK12345» som har takstgruppe «electric» har kjørt inn på parkeringsområdet:

```
const arrival = carpark.addCar("EK12345", "electric");
```

Returverdi er følgende objekt:

```
{
  regno: "EK12345",
  taxgroup: "electric",
  arrival: 1637065716428
}
```

Egenskapene *regno* og *taxgroup* viser parameterverdiene som ble brukt i metodekallet. Egenskapen *arrival* er tidspunktet for når bilen ble registrert inn, representert ved antall millisekunder siden 1. januar 1970.

- *removeCar(regno)*: Metoden brukes når bil med registreringsnummer *regno* forlater parkeringsområdet.

Parametere:

- Inn-parameter: **String**
- Returverdi: **Object**

Metoden returnerer et objekt, eller *null*. Returverdi er *null* hvis bilen ikke var registrert inn på parkeringsområdet.

Kodeeksempelet under fjerner bilen «EK12345» fra *carpark*:

```
const departure = carpark.removeCar("EK12345");
```

Returverdi er følgende objekt:

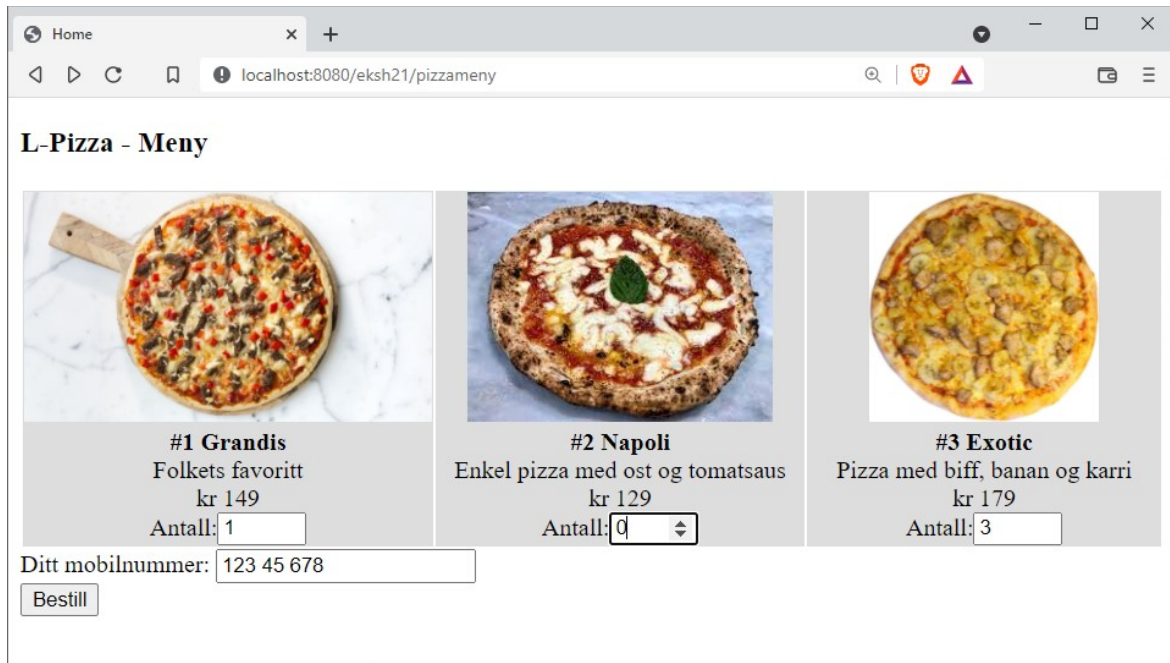
```
{
  regno: "EK12345",
  taxgroup: "electric",
  arrival: 1637065716428,
  departure: 1637070036428,
  cost: 25
}
```

Egenskapene *regno*, *taxgroup* og *arrival* er som for *addCar*. Egenskapen *departure* er tidspunktet for når bilen forlot parkeringsområdet, representert ved antall millisekunder siden 1. januar 1970. Egenskapen *cost* er parkeringsavgiften for denne parkeringen.

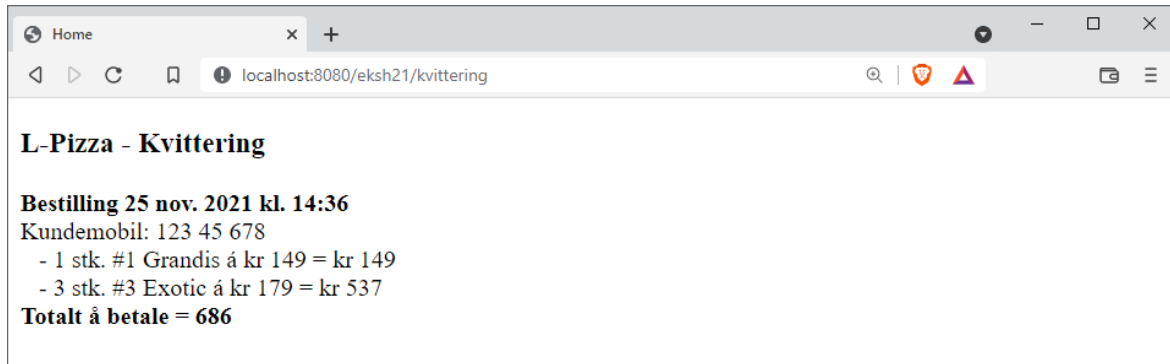
Oppgave: Skriv JavaScript-koden for **Parking** i samsvar med teksten over.

Oppgave 3 (40% ~ 96 minutter) – Webapplikasjoner, tjenerside

Dere skal lage litt av en løsning for online bestilling hos en pizza-restaurant.



Bilde 3.1 - /pizzameny



Bilde 3.2 - /kvittering

Vi begynner med en forklaring av hvordan applikasjonen er bygget opp.

Informasjonen om pizzaene i menyen er lagret i en databasetabell **pizza**,

nr [PK] integer	navn character varying	beskrivelse character varying	pris integer	bildefil character varying
1	Grandis	Folkets favoritt	149	grandis.jpg
2	Napoli	Enkel pizza med ost og tomatsaus	129	napoli.jfif
3	Exotic	Pizza med biff, banan og karri	179	exotic.jpg

Bilde 3.3 - Databasetabell med eksempeldata

Vi har tilsvarende JPA entitetsklasse i Java kalt **Pizza**.

Navn på bildefil refererer til et bilde som ligger i applikasjonen under **.../webapp/bilder/**

Vi har en hjelpeklasse (**@Stateless EJB**) **PizzaDAO** som vi kan bruke til å lese inn data fra databasen til applikasjonen. Metoden som kan brukes heter **List<Pizza> hentAllePizzaer()**.

Applikasjonen skal hente inn dataene ved/før første gangs bruk og deretter holde dem i minnet slik at de er tilgjengelig for alle deler av applikasjonen, dvs. alle Servleter og alle JSP-er, på en grei måte. (*Hint: Hvilket scope er egnet til dette?*)

Pizzameny-siden skal bygges opp på grunnlag av disse dataene. For enkelhets skyld kan vi si at siden viser en tabell av pizzaer med én rad og n kolonner. Se Bilde 3.1.

Bestilling gjøres ved at man velger antall av de ulike pizzaene, gir inn mobilnummer for referanse og trykker Bestill (~ **POST /pizzameny**).

Det skal da lages et **Bestilling**-objekt, dette skal lagres i databasen via **PizzaDAO** sin **void lagreBestilling(Bestilling b)**. Deretter skal kvitteringssiden vises som i Bilde 3.2.

Et **Bestilling**-objekt inneholder følgende data

- **tidspunkt for bestillingen** (LocalDateTime)
- **kundens mobil** (String)
- **antall av de ulike pizzaene** (f.eks. Map av <Pizza, Integer>). *Grunnen til at hele Pizzaen foreslås brukt som nøkkel er at man da enkelt kan få alle pizzaene ut ved å bruke metoden **Map.keySet()**. Det er kanskje mer/like ryddig å bruke pizza.nr som nøkkel. Velg den måten du foretrekker av disse.*

og konstruktører og metoder **etter behov**.

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 hjelpeklassene**, **elegant kode**, osv ...

Løsningen trenger ikke å være robust mot tekniske databasefeil. Du kan anta at det hentes minst ett pizza-objekt fra databasen når du spør etter det.

Løsningen trenger ikke å være robust mot «hacking», dvs. utilsiktet bruk. Du kan anta at nødvendig validering er gjort i nettleseren via HTML eller JavaScript. Altså at antall som skal bestilles av de ulike pizzaene er 0 eller et positivt heltall av rimelig størrelse og at mobilnummer er noe som godtas av applikasjonen.

Oppgaver

- (15%) Skriv **PizzamenyServlet** som mappes til URLen **/pizzameny**. Tilhørende view er **pizzameny.jsp**.
- (10%) Skriv **pizzameny.jsp**. *Tips: Parameternavn for antall av de ulike pizzaene kan kanskje være en kombinasjon av noe fast + pizza-nr.*
- (5%) Skriv **kvittering.jsp**. *Tips1: Siden Bestilling er en nøstet struktur (inneholder et Map med antall av hver pizza) kan det kanskje være lurt/nødvendig å lagre detaljer underveis i page-attributter for ikke å få alt for komplekse EL-uttrykk. Tips2: Å hente en verdi fra et map m kan gjøres med m[key].*
- (5%) List opp alle konstruktører og metoder i klassen **Bestilling** som du har brukt i løsningen. Du trenger ikke å vise implementasjon, kun **returtype**, **metodenavn** og evt. **parametre m/typer**. Skriv gjerne en kort kommentar om det er uklart hva metoden gjør.
- (5%) La oss nå tenke oss at vi ønsker å kunne tilby denne pizzabestillings-appen på flere språk, og vi ønsker å bruke request-headeren **Accept-Language** til å bestemme hvilket språk vi skal velge. La oss videre tenke oss at vi har en ferdig hjelpeklasse **SpraakUtil** med hjelpemetoden **finnBesteSprak** vi kan bruke til dette:

```
public class SpraakUtil {
    public static String finnBesteSprak(HttpServletRequest request) { ... }
}
```

F.eks. skal en request med headeren **Accept-Language: fr, en;q=0.8, de;q=0.7** returnere **"fr"** som som foretrukket språk.

Hvordan kan man utføre **enhetstesting** for å teste at **SpraakUtil.finnBesteSprak(...)** med headeren i eksemplet over gir **"fr"** som svar. For å få full score må du skrive en fungerende test.

Oppgave 4 (10% ~ 24 minutter) – Tråder

Vi tar utgangspunkt i programmet under. Her bruker vi et stoppeklokke-objekt til å ta tiden, og vi bruker JOptionPane til å starte og stoppe stoppeklokken.

```

public class StoppeklokkeMain {
    public static void main(String[] args) throws InterruptedException {
        Stopwatch stoppeklokke = new Stopwatch();

        JOptionPane.showConfirmDialog(...);
        stoppeklokke.start();

        JOptionPane.showConfirmDialog(...);
        stoppeklokke.stop();

        System.out.println("\nSluttid: " + stoppeklokke.formatTime());
    }
}

```

Start

Trykk for å starte tidtaking

OK

Stopp

Trykk for å stoppe tidtaking

OK

Sluttid: 00:00:58.245

Vi har lyst å fikse dette programmet slik at det i tillegg, mens stoppeklokken går, kontinuerlig (f.eks. hvert 10. millisekund) viser tiden så langt. (NB! Dette må skje samtidig som vi venter på at bruker skal stoppe tidtakingen.)

Ved å bruke **System.out.print("\r" + det som skal skrives ut)** i en løkke kan vi skrive på samme linje i konsollet om igjen og om igjen uten at vi hopper til neste linje. (Antar at "\r" (=return) hopper til starten av linjen uten å hoppe ned slik "\n" (=newline) gjør.)

Oppgave: Gjør de nødvendige endringer i programmet + lag evt. nye interfaces og klasser du trenger for å få dette til. *Tips: Det er sikkert lurt å lage en trådklasse.*

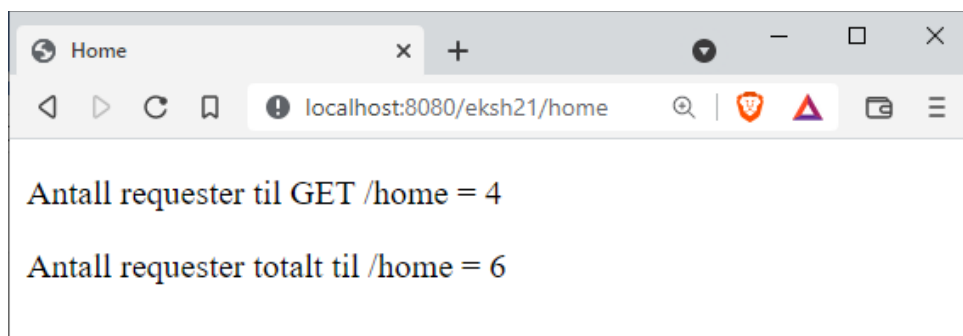
Oppgave 5 (10% ~ 24 minutter) – Web og tråder

Som dere vet vil en Java web-container (f.eks. Tomcat) opprette en ny tråd for hver request og kjøre `doGet()` eller `doPost()` i denne tråden. Tenk at vi har en applikasjon med én Servlet (HomeServlet) med `doGet()` og `doPost()`.

Vi ønsker å holde orden på

1. Hvor mange requester det har vært til GET /home (`doGet()` i HomeServlet) siden oppstart
2. Hvor mange requester det har vært totalt (både GET og POST /home) siden oppstart

Begge tellerne skal kunne hentes frem i tilhørende JSP-side med expression language.



Oppgave: Skriv en løsning for hvordan disse to tellerne kan implementeres slik at de teller korrekt og på en trådsikker måte. Altså skriv det som er relevant i HomeServlet.

EKSAMENSOPPGÅVE

Emnekode: DAT108

Emnenamn: Programmering og webapplikasjoner

Klasse: 2. klasse DATA / INF

Dato: 9. desember 2021

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 (20% ~ 48 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.

a) Tenk at du har ei liste av hundena slik:

```
List<String> hunder = List.of("Fido", "Buster", "Colin");
```

Ved å bruka **forEach(c)**-metoden på lista med ulike parametarar **c** kan du laga 3 ulike utskrifter, slik:

Fido Buster Colin	*Fido* *Buster* *Colin*	odiF retsuB niloC
-------------------------	-------------------------------	-------------------------

Skriv dei 3 ulike variantane av parameteren **c** og lagre dei i variablane **printPlain**, **printMedStjerner** og **printBaklengs**. Hugs å få med datatypen til variablane. *Tips til baklengs: `new StringBuilder(s).reverse()` vil snu strengen s.*

I dei neste deloppgåvene skal vi sjå på ei liste av bøker. Ei Bok har dissa eigenskapane:

```
class Bok {
    public String tittel;
    public int aar;
    public List<String> forfattere; //Kvar forfattar er ein enkel String
    ... konstruktører og metoder ...
}
```

Tenk så at vi t.d. har ei liste med bøker slik:

```
List<Bok> boker = List.of(
    new Bok("Core Java Volume I", 2022, List.of("Cay Horstmann")),
    new Bok("Effective Java", 2017, List.of("Cay Horstmann")),
    ...
    new Bok("Head First Java", 2005,
        List.of("Kathy Sierra", "Bert Bates")),
    new Bok("Java Concurrency in Practice", 2006,
        List.of("Brian Goetz", "Tim Peierls", "Joshua Bloch"))
);
```

Du skal no løysa nokre oppgåver ved å bruka streams og lambda-uttrykk.

- b) Skriv ei setning som lagar ei ny liste av alle bøker som er utgitt i 2015 eller tidligare
- c) Skriv ei setning som lagar ei liste av alle boktitlar som inneheld ordet Java
- d) Skriv ei setning som lagar ei liste av alle forfattarane utan duplikat
- e) Utvid d) med at forfattarane blir sorterte på etternamn ved å bruke `Stream<T> sin sorted(Comparator<? super T> comparator)` der comparatoren er tilordna ein variabel kalla **paaEtternavn** før bruk. Tips: `s.substring(s.lastIndexOf(" "))` gir etternamnet for namnet `s`.

Oppgave 2 (20% ~ 48 minutt) – JavaScript

- a) HTML taggen *script* kan verte nytta saman med eit attributt *defer*.
- Kva er konsekvensen av *defer* attributtet. Kan *defer* ha noko å seie for ytinga til webapplikasjonen? Grunnge svaret.
 - Kor i eit webdokument bør du plassere ein script tagg med *defer*? Grunnge svaret.
 - Kor i eit webdokument bør du plassere ein script tagg utan *defer*? Grunnge svaret.
- b) For å modifisere DOM-strukturen til eit webdokument har vi til dømes eigenskapane *innerHTML*, *textContent* og *innerText*, og metoden *insertAdjacentHTML*.
- Gje døme på når det er riktig å nytte *innerHTML* og *insertAdjacentHTML*. Grunnge svaret.
 - Gje døme på situasjonar der *innerHTML* og *insertAdjacentHTML* ikkje må bli nytta. Grunnge svaret.
 - Gje døme på når det er riktig å nytte *textContent* og *innerText*. Grunnge svaret.
- c) Opprett ei JavaScript klasse **Parking**. Instansar av klassa må kunne administrere eit parkeringsområde og parkeringsavgift for bilar på området.

Eit parkeringsområde har eit gjeve tal for plassar til bilar, og kvar bil må høyre til ei takstgruppe. Klassa har metodane *addCar* og *removeCar* for å registrere at biler køyrer inn og ut av parkeringsområdet.

I døme under lagar kode lina ein instans av **Parking**:

```
const carpark= new Parking(50, { electric: 5, normal: 30 });
```

Første argument til konstruktøren er tal på parkeringsplassar på parkeringsområdet, og andre argument er eit objekt med takstgrupper og parkeringsavgift per starta 15 minutt intervall. I døme kostar det 5 kroner per starta 15 minutt intervall for bilar i takstgruppe *electric* og 30 for bilar i takstgruppe *normal*.

For bilar som forlèt området innan 10 minutt blir det ikkje regna parkeringsavgift.

Klassa **Parking** har følgjande metodar:

- *addCar(regno,taxgroup)*: Metoden registrerer at bil med registreringsnummer *regno* som er i takstgruppe *taxgroup* har køyrd inn på parkeringsområdet.

Parametrar:

- Inn-parametrar: **String, String**
- Returverde: **Object**

Metoden returnerer eit objekt, eller *null*. Returverde *null* tyder at registrering av bil feila. Grunnar for returverde *null* kan være at parkeringsområdet er fullt, eller at takstgruppa *taxgroup* ikkje finnast.

Kode lina i dømme under registrerer at bilen «EK12345» som har takstgruppe «electric» har kjørd inn på parkeringsområdet:

```
const arrival = carpark.addCar("EK12345", "electric");
```

Returverde er følgjande objekt:

```
{
  regno: "EK12345",
  taxgroup: "electric",
  arrival: 1637065716428
}
```

Egenskapane *regno* og *taxgroup* viser parameterverde som blei nytta i metodekallet. Egenskapen *arrival* er tidspunktet for når bilen blei registrert inn, representert ved tal på millisekund sidan 1. januar 1970.

- *removeCar(regno)*: Metoden bli nytta når bil med registreringsnummer *regno* forlèt parkeringsområdet.

Parametrar:

- Inn-parameter: **String**
- Returverde: **Object**

Metoden returnerer eit objekt, eller *null*. Returverde er *null* om bilen ikkje var registrert på parkeringsområdet.

Kode lina i dømme under fjernar bilen «EK12345» frå *carpark*:

```
const departure = carpark.removeCar("EK12345");
```

Returverde er følgjande objekt:

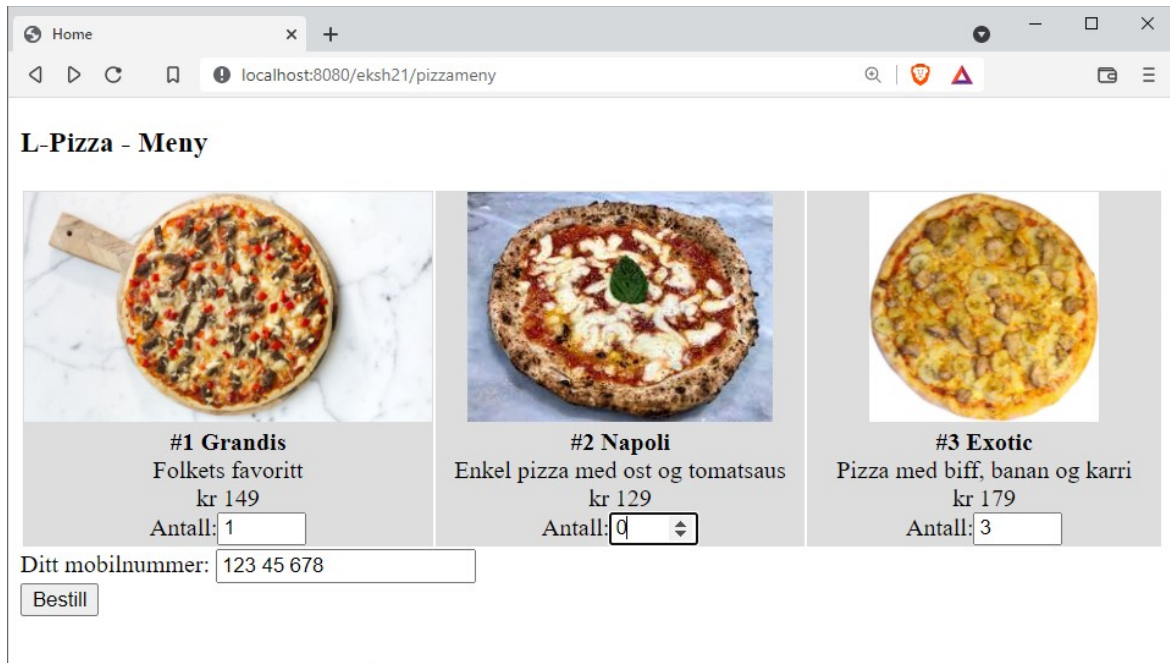
```
{
  regno: "EK12345",
  taxgroup: "electric",
  arrival: 1637065716428,
  departure: 1637070036428,
  cost: 25
}
```

Egenskapane *regno*, *taxgroup* og *arrival* er som for *addCar*. Egenskapen *departure* er tidspunktet for når bilen forlèt parkeringsområdet, representert ved tal på millisekund sidan 1. januar 1970. Egenskapen *cost* er parkeringsavgifta for denne parkeringa.

Oppgåve: Skriv JavaScript-koden for **Parking** i samsvar med teksten over.

Oppgave 3 (40% ~ 96 minutt) – Webapplikasjoner, tenarside

De skal lage litt av ei løysing for online bestilling hos ein pizza-restaurant.

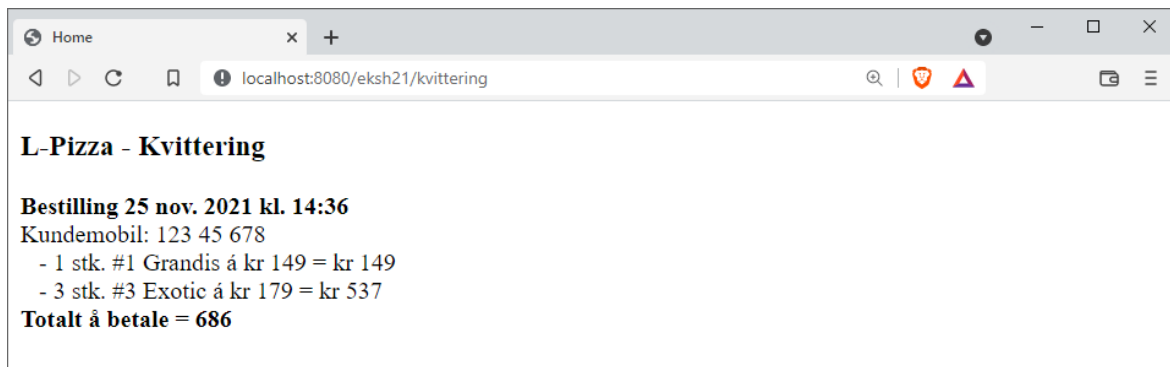


L-Pizza - Meny

#1 Grandis	#2 Napoli	#3 Exotic
Folkets favoritt	Enkel pizza med ost og tomat saus	Pizza med biff, banan og karri
kr 149	kr 129	kr 179
Antall: <input type="text" value="1"/>	Antall: <input type="text" value="0"/>	Antall: <input type="text" value="3"/>

Ditt mobilnummer:

Bilde 3.1 - /pizzameny



L-Pizza - Kvittering

Bestilling 25 nov. 2021 kl. 14:36

Kundemobil: 123 45 678

- 1 stk. #1 Grandis á kr 149 = kr 149
- 3 stk. #3 Exotic á kr 179 = kr 537

Totalt å betale = 686

Bilde 3.2 - /kvittering

Vi byrjar med ei forklaring av korleis applikasjonen er bygd opp.

Informasjonen om pizzaene i menyen er lagra i ein databasetabell **pizza**,

nr [PK] integer	navn character varying	beskrivelse character varying	pris integer	bildefil character varying
1	Grandis	Folkets favoritt	149	grandis.jpg
2	Napoli	Enkel pizza med ost og tomatsaus	129	napoli.jfif
3	Exotic	Pizza med biff, banan og karri	179	exotic.jpg

Bilde 3.3 - Databasetabell med eksempeldata

Vi har tilsvarande JPA entitetsklasse i Java kalla **Pizza**.

Navn på bildefil refererer til eit bilde som ligg i applikasjonen under **.../webapp/bilder/**

Vi har ein hjelpeklasse (@Stateless EJB) **PizzaDAO** som vi kan bruka til å lesa inn data frå databasen til applikasjonen. Metoden som kan brukast heiter **List<Pizza> hentAllePizzaer()**.

Applikasjonen skal henta inn dataa ved/før første gongs bruk og deretter halda dei i minnet slik at dei er tilgjengeleg for alle delar av applikasjonen, dvs. alle Servleter og alle JSP-er, på ein grei måte. (*Hint: Kva scope er eigna til dette?*)

Pizzameny-sida skal byggast opp på grunnlag av desse dataa. For enkelhets skuld kan vi seia at sidan viser ein tabell av pizzaer med éi rad og n kolonnar. Sjå Bilde 3.1.

Bestilling blir gjort ved at ein vel tal av de ulike pizzaene, gir inn mobilnummer for referanse og trykker Bestill (~ **POST /pizzameny**).

Det skal då lagast eit **Bestilling**-objekt, dette skal lagrast i databasen via **PizzaDAO** sin **void lagreBestilling(Bestilling b)**. Deretter skal kvitteringssiden visast som i Bilde 3.2.

Eit **Bestilling**-objekt inneheld følgjande data

- **tidspunkt for bestillingen** (LocalDateTime)
- **mobilen til kunden** (String)
- **tal av dei ulike pizzaene** (f.eks. Map av <Pizza, Integer>). *Grunnen til at heile Pizzaen blir foreslått brukt som nøkkel er at ein då enkelt kan få alle pizzaene ut ved å bruka metoden **Map.keySet()**. Det er kanskje meir/like ryddig å bruka pizza.nr som nøkkel. Vel den måten du føretrekker av desse.*

og konstruktører og metodar **etter behov**.

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 brukerinput, **unntakshandtering**, **god bruk av hjelpeklassane**, **elegant kode**, osv ...

Løysinga treng ikkje å vera robust mot tekniske databasefeil. Du kan anta at det blir henta minst eitt pizza-objekt frå databasen når du spør etter det.

Løysinga treng ikkje å være robust mot «hacking», dvs. utilsikta bruk. Du kan anta at nødvendig validering er gjort i nettlesaren via HTML eller JavaScript. Altså at tal som skal bestillast av dei ulike pizzaene er 0 eller eit positivt heltall av rimeleg storleik og at mobilnummer er noko som blir godtatt av applikasjonen.

Oppgåver

- (15%) Skriv **PizzamenyServlet** som mappes til URLen **/pizzameny**. Tilhøyrande view er **pizzameny.jsp**.
- (10%) Skriv **pizzameny.jsp**. *Tips: Parameternamn for tal av dei ulike pizzaene kan kanskje vera ein kombinasjon av noko fast + pizza-nr.*
- (5%) Skriv **kvittering.jsp**. *Tips1: Sidan Bestilling er ein nøsta struktur (inneheld eit Map med tal av kvar pizza) kan det kanskje være lurt/nødvendig å lagra detaljar undervegs i page-attributter for ikkje å få alt for komplekse EL-uttrykk. Tips2: Å henta ein verdi frå eit map m kan gjerast med `m[key]`.*
- (5%) List opp alle konstruktører og metodar i klassen **Bestilling** som du har brukt i løysinga. Du treng ikkje å vise implementasjon, berre **returtype**, **metodenamn** og evt. **parametrar m/typar**. Skriv gjerne ein kort kommentar om det er uklart kva metoden gjer.
- (5%) La oss no tenka oss at vi ønsker å kunna tilby denne pizzabestillings-appen på fleire språk, og vi ønsker å bruka request-headeren **Accept-Language** til å bestemma kva språk vi skal velja. La oss vidare tenka oss at vi har ein ferdig hjelpeklasse **SpraakUtil** med hjelpemetoden **finnBesteSpraak** vi kan bruka til dette:

```
public class SprakUtil {
    public static String finnBesteSpraak(HttpServletRequest request) { ... }
}
```

T.d. skal ein request med headeren **Accept-Language: fr, en;q=0.8, de;q=0.7** returnera **"fr"** som som føretrekt språk.

Korleis kan ein utføra **enhetstesting** for å testa at **SpraakUtil.finnBesteSpraak(..)** med headeren i dømet over gir **"fr"** som svar. For å få full score må du skrive ein fungerande test.

Oppgave 4 (10% ~ 24 minutt) – Trådar

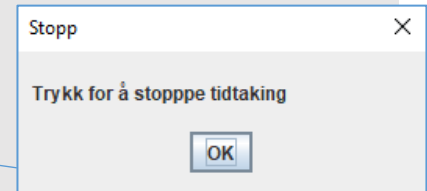
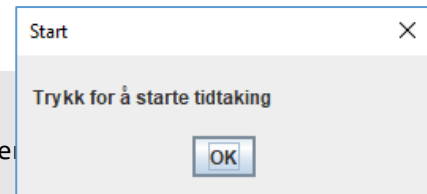
Vi tar utgangspunkt i programmet under. Her bruker vi eit stoppeklokke-objekt til å ta tida, og vi bruker JOptionPane til å starta og stoppa stoppeklokken.

```
public class StoppeklokkeMain {
    public static void main(String[] args) throws InterruptedException {
        Stopwatch stoppeklokke = new Stopwatch();

        JOptionPane.showConfirmDialog(...);
        stoppeklokke.start();

        JOptionPane.showConfirmDialog(...);
        stoppeklokke.stop();

        System.out.println("\nSluttid: " + stoppeklokke.formatTime());
    }
}
```



Sluttid: 00:00:58.245

Vi har lyst å fiksa dette programmet slik at det i tillegg, medan stoppeklokken går, kontinuerleg (t.d. kvart 10. millisekund) viser tida så langt. (NB! Dette må skje samtidig som vi ventar på at brukar skal stoppa tidtakinga.)

Ved å bruka **System.out.print("\r" + det som skal skrivast ut)** i ein løkke kan vi skriva på same linje i konsollet om igjen og om igjen utan at vi hoppar til neste linje. (Antar at "\r" (=return) hoppar til starten av linja utan å hoppa ned slik "\n" (=newline) gjer.)

Oppgave: Gjer dei nødvendige endringane i programmet + lag evt. nye interfaces og klassar du treng for å få dette til. *Tips: Det er sikkert lurt å laga ein trådklasse.*

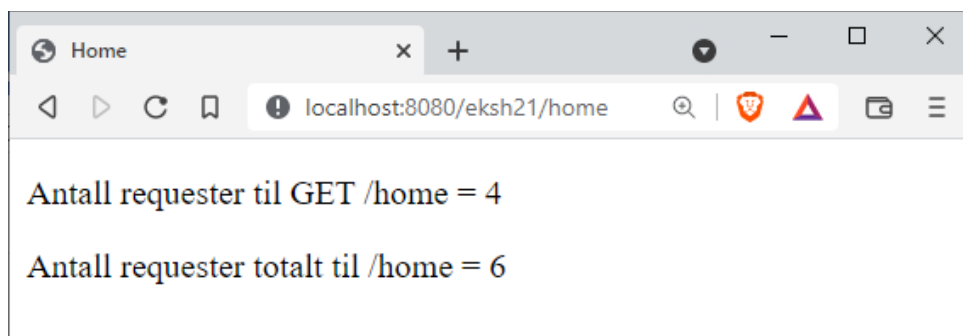
Oppgave 5 (10% ~ 24 minutt) – Web og trådar

Som de veit vil ein Java web-container (t.d. Tomcat) oppretta ein ny tråd for kvar request og køyra `doGet()` eller `doPost()` i denne tråden. Tenk at vi har ein applikasjon med éin Servlet (HomeServlet) med `doGet()` og `doPost()`.

Vi ønsker å halda orden på

1. Kor mange requester det har vore til GET `/home` (`doGet()` i HomeServlet) sidan oppstart
2. Kor mange requester det har vore totalt (både GET og POST `/home`) sidan oppstart

Begge teljarane skal kunna hentast fram i tilhøyrande JSP-side med expression language.



Oppgave: Skriv ei løysing for korleis desse to teljarane kan implementerast slik at de tel korrekt og på ein trådsikker måte. Altså skriv det som er relevant i HomeServlet.