DAT108 Oblig1 h22 - Tråder

Sist oppdatert av Lars-Petter Helland 05.09.2022

Innleveringsfrist er søndag 11. september. Vi har som mål å rette innleveringene og godkjenne innen 2 uker etter frist.

Øvingen leveres ut i 2 omganger:

- Oppg 1 og 2 leveres ut fredag 26. august
- Ny versjon med Oppg 3 og 4 leveres ut fredag 2. september

Øvingen skal gjennomføres i grupper på **2-4** studenter.

Dere danner selv grupper i Canvas ved å registrere dere i en av de 80 forhåndsdefinerte gruppene «DAT108 Oblig1 gruppe x». Legg helst til alle studentene i gruppen samtidig. (Gå inn på «Personer» | «DAT108 Oblig1 gruppe», og legg til dere selv i en tom gruppe).

Grupper større enn 4 studenter kan kun godkjennes etter avtale (, men vi foretrekker det fremfor at dere deler dere og leverer to identiske løsninger).

Det er i utgangspunktet ikke anledning til å levere alene. Dere oppfordres til å finne samarbeidspartnere selv, men vi vil hjelpe til ved behov. Grupper med 1 student kan også kun godkjennes etter avtale.

Send en melding til lærer (Lars-Petter) ved ønsker gruppestr. utenfor 2-4.

Innleveringen er en zip i Canvas. Denne skal inneholde:

1. Et pdf-dokument som inneholder:

    a) en liste av hvem som er med i gruppen (for å unngå gratispassasjerer som melder seg inn i gruppe uten at det er avtalt).

    b) Skjermutskrift fra kjøringer av programmene, slik at vi kan se at de virker.

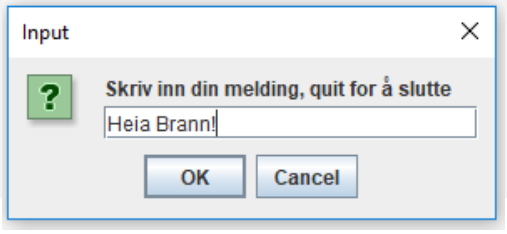
2. Løsning på oppgavene: **Ett** Eclipse-prosjekt med komplette kjørbare løsninger. Løsningen på de ulike oppgavene kan f.eks. ligge i ulike pakker (...oppgave1, ...oppgave2, osv) i dette prosjektet.

Oppgave 1 – System.out.println og JOptionPane [Kan gjøres etter F02]

Dere skal lage et lite program der vi har én tråd som skriver ut en linje på skjermen hvert 3. sekund, og én tråd som gjentatte ganger lar brukeren taste inn via JOptionPane hva som skal skrives ut. Det skal være en egen «kommando» (f.eks. quit) for å avslutte hele programmet.

Nedenfor er et eksempel på kjøring der programmet først skriver ut standardmeldingen «Hallo verden!», og deretter skriver ut det som bruker har tastet inn («Heia Brann!»). Brukeren skal ha anledning til å taste inn nye meldinger så mange ganger han vil samtidig som forrige melding skrives ut om igjen og om igjen på skjermen.

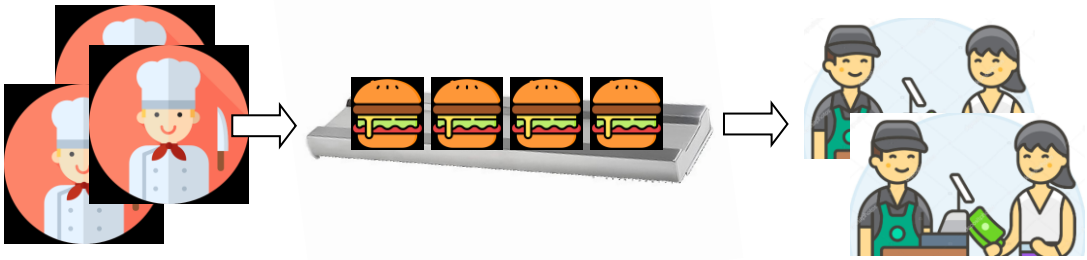
```
Hallo verden!
Hallo verden!
Hallo verden!
Hallo verden!
Hallo verden!
Heia Brann!
Heia Brann!
Heia Brann!
Heia Brann!

...
```



Oppgave 2 – Hamburgersjappe [Kan gjøres etter F03]

Dere skal lage et lite program som simulerer en hamburger-sjappe.

Hamburger-sjappen har et **brett** (kø) der ferdige **hamburger**e legges på av **kokk**en(e) etter hvert som de er ferdige og tas av av **servitør**en(e) etter hvert som kunder kjøper en hamburger.



Utskriften fra en simulering skal se ut ca. slik:

```
I denne simuleringen har vi
       3 kokker [Anne, Erik, Knut]
       2 servitører [Mia, Per]
       Kapasiteten til brettet er 4 hamburgere.
Vi starter ...

Anne (kokk) legger på hamburger ◖1◗. Brett: [◖1◗]
Mia (servitør) tar av hamburger ◖1◗. Brett: []
Per (servitør) ønsker å ta hamburger, men brett tomt. Venter!
Anne (kokk) legger på hamburger ◖2◗. Brett: [◖2◗]
Per (servitør) tar av hamburger ◖2◗. Brett: []
Per (servitør) ønsker å ta hamburger, men brett tomt. Venter!
Knut (kokk) legger på hamburger ◖3◗. Brett: [◖3◗]
Per (servitør) tar av hamburger ◖3◗. Brett: []
Erik (kokk) legger på hamburger ◖4◗. Brett: [◖4◗]
Per (servitør) tar av hamburger ◖4◗. Brett: []
Knut (kokk) legger på hamburger ◖5◗. Brett: [◖5◗]
Erik (kokk) legger på hamburger ◖6◗. Brett: [◖5◗, ◖6◗]
Mia (servitør) tar av hamburger ◖5◗. Brett: [◖6◗]
Anne (kokk) legger på hamburger ◖7◗. Brett: [◖6◗, ◖7◗]
Knut (kokk) legger på hamburger ◖8◗. Brett: [◖6◗, ◖7◗, ◖8◗]
Per (servitør) tar av hamburger ◖6◗. Brett: [◖7◗, ◖8◗]
Anne (kokk) legger på hamburger ◖9◗. Brett: [◖7◗, ◖8◗, ◖9◗]
Mia (servitør) tar av hamburger ◖7◗. Brett: [◖8◗, ◖9◗]
Erik (kokk) legger på hamburger ◖10◗. Brett: [◖8◗, ◖9◗, ◖10◗]
Knut (kokk) legger på hamburger ◖11◗. Brett: [◖8◗, ◖9◗, ◖10◗, ◖11◗]
Per (servitør) tar av hamburger ◖8◗. Brett: [◖9◗, ◖10◗, ◖11◗]
Erik (kokk) legger på hamburger ◖12◗. Brett: [◖9◗, ◖10◗, ◖11◗, ◖12◗]
Mia (servitør) tar av hamburger ◖9◗. Brett: [◖10◗, ◖11◗, ◖12◗]
Per (servitør) tar av hamburger ◖10◗. Brett: [◖11◗, ◖12◗]
Anne (kokk) legger på hamburger ◖13◗. Brett: [◖11◗, ◖12◗, ◖13◗]
Mia (servitør) tar av hamburger ◖11◗. Brett: [◖12◗, ◖13◗]
Erik (kokk) legger på hamburger ◖14◗. Brett: [◖12◗, ◖13◗, ◖14◗]
Knut (kokk) legger på hamburger ◖15◗. Brett: [◖12◗, ◖13◗, ◖14◗, ◖15◗]
Anne (kokk) klar med hamburger, men brett fullt. Venter!
Per (servitør) tar av hamburger ◖12◗. Brett: [◖13◗, ◖14◗, ◖15◗]
Anne (kokk) legger på hamburger ◖16◗. Brett: [◖13◗, ◖14◗, ◖15◗, ◖16◗]

... (osv) ...
```

Krav til programmet

- Både **Hamburger**, **HamburgerBrett**, **Kokk** og **Servitor** skal være klasser i programmet slik det er naturlig. Kokker og servitører er tråder, mens brettet er en delt ressurs som både kokker og servitører forholder seg til.
- De ulike objektene i programmet skal samarbeide via referanser til hverandre, ikke gjennom globale variabler.
- Programmet må være trådsikkert slik at det ikke blir rot i tellingen og i køen av hamburgere. Kokker og servitører må vente hvis det er nødvendig, ikke bare kjøre videre.
- Hvis brettet er tomt, må servitører vente til det er lagt en ny hamburger på, og hvis brettet er fullt, må kokker vente til det er tatt en hamburger av.
- Det er meningen at dere skal bruke tråd-primitivene **synchronized**, **wait**, **notify** og **notifyAll** til å løse oppgaven. Dere kan ikke bruke klasser fra java.util.concurrent-API-et, f.eks. Lock og BlockingQueue.
- Tiden det tar å lage og bestille en hamburger skal være (random) mellom 2 og 6 sek.

main() kan f.eks. se slik ut:

```
public static void main(String... blablabla) {

      final String[] kokker = {"Anne", "Erik", "Knut"};
      final String[] servitorer = {"Mia", "Per"};
      final int KAPASITET = 4;

      skrivUtHeader(kokker, servitorer, KAPASITET);

      HamburgerBrett brett = new HamburgerBrett(KAPASITET);

      for (String navn : kokker) {
            new Kokk(brett, navn).start();
      }
      for (String navn : servitorer) {
            new Servitor(brett, navn).start();
      }
}
```

Oppgave 3 – Hamburgere med BlockingQueue

Samme oppgave som i Oppgave 2, men her kan/skal dere bruke java.util.concurrent.BlockingQueue i stedet for synchronized, wait, notify.

Merk: Ikke mulig med så detaljert utskrift over hva som foregår (venting og sånn). Utskriften kan f.eks. være slik:

```
I denne simuleringen har vi
       3 kokker [Anne, Erik, Knut]
       2 servitører [Mia, Per]
       Kapasiteten til brettet er 4 hamburgere.
Vi starter ...

Anne (kokk) legger på hamburger ◖1◗. Brett: [◖1◗]
Mia (servitør) tar av hamburger ◖1◗. Brett: []
Anne (kokk) legger på hamburger ◖2◗. Brett: [◖2◗]
Per (servitør) tar av hamburger ◖2◗. Brett: []
Knut (kokk) legger på hamburger ◖3◗. Brett: [◖3◗]
Per (servitør) tar av hamburger ◖3◗. Brett: []
Erik (kokk) legger på hamburger ◖4◗. Brett: [◖4◗]
Per (servitør) tar av hamburger ◖4◗. Brett: []
Knut (kokk) legger på hamburger ◖5◗. Brett: [◖5◗]
Erik (kokk) legger på hamburger ◖6◗. Brett: [◖5◗, ◖6◗]
Mia (servitør) tar av hamburger ◖5◗. Brett: [◖6◗]
Anne (kokk) legger på hamburger ◖7◗. Brett: [◖6◗, ◖7◗]
Knut (kokk) legger på hamburger ◖8◗. Brett: [◖6◗, ◖7◗, ◖8◗]

... (osv) ...
```

Oppgave 4 – UTGÅR