TDT4186 Operativsystemer

Øvingsforelesning 1

Agenda

- Generelt om øvingsopplegget
- Introduksjon til tråder og synkronisering
- Presentasjon av praktisk øving P1

Øvingsstab

- Undass:
  - Thomas Aven (thomaav@stud.ntnu.no) (bruk emne: «[TDT4186]…»)
- Studasser:
  - Tinus Sola Flagstad
  - Sondre Havellen
  - Øystein Krogstie
  - Aksel Hauge Slettemark
  - Eivind Standal

- Piazza

Øvingene

- 1 praktisk (**obligatorisk**) – kap. 5 + 6
  - Synkronisering av kommunikasjon mellom prosesser
  - Kap. 5: Forhold mellom prosesser: **mutex**, **semaphores**, **monitors**
  - Kap. 6: Problemer med synkronisering: **deadlock**, **starvation**

- 2 teoretiske (frivillige) – kap. 7 + 8 og 9 + 10
  - Kap. 7+8: Bruk av lager, virtuelt minne, lokalitetsprinsippet
  - Kap. 9+10: Kjøring av prosesser, multiprosessor
  - Oppgaver fra læreboken

Innlevering

- Individuelt arbeid.

- Leveres på Blackboard innen fristen.
  - Frist mandag 18. mars 23:59

- Demonstrering av praktisk øving på sal til studass
  - Frist fredag 22. mars
  - Kommer påmelding for å fordele demonstrasjoner jevnt

Saltider

- **Øvingssal**: A3-125 i Realfagbygget

- Mandager 1600-2000 – Ukene 10-12
- Tirsdager 1600-2000 – Ukene 10-12
- Torsdager 1600-2000 – Ukene 9-12
- Fredager 1400-1800 – Ukene 9-12

- Brukes til å få godkjent øvinger eller spørre om hjelp

Semesterplan

Uke	Innhold	Kommentar
09 (28.02)	Orientering om øvingsopplegget Introduksjon til tråder og synkronisering Presentasjon av praktisk øving P1 (Oblig.)	P1: Forhold mellom prosesser - Kap. 5 +6
10 (07.03)	Diskusjon av praktisk øving P1 Presentasjon av teoretisk øving TA (Frivil.)	TA: Bruk av lager - Kap. 7 & 8
11 (14.03)	Diskusjon av praktisk øving P1 Diskusjon av teoretisk øving TA Presentasjon av teoretisk øving TB (Frivil.)	TB: Kjøring av prosesser - Kap. 9 & 10
12 (21.03)	Diskusjon av teoretisk øving TA Diskusjon av teoretisk øving TB	Innleveringsfrist for P1: *Mandag i uke 12 (dvs. 18.03) - Oblig.* Godkjenningsfrist for P1: *Fredag i uke 12 (dvs. 22.03) - Oblig.*
13 (28.03)	Gjennomgang av praktisk øving P1	
14 (04.04)	Gjennomgang av teoretisk øving TA Gjennomgang av teoretisk øving TB	


spørsmål?

Agenda

- Generelt om øvingsopplegget
- Introduksjon til tråder og synkronisering
- Presentasjon av praktisk øving P1

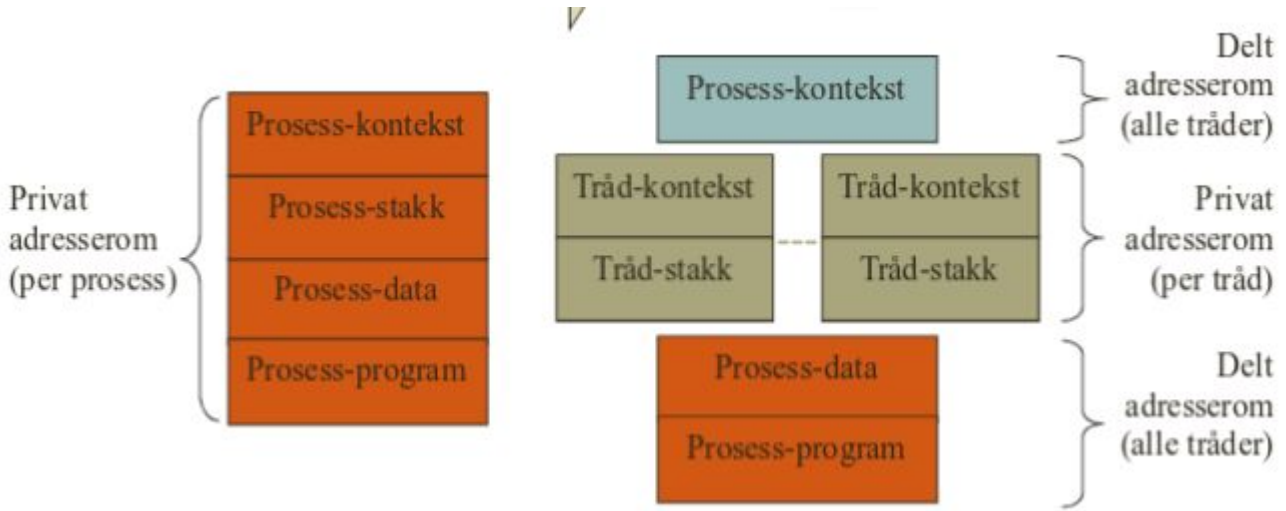
Prosesser

- En instans av et kjørende program (under utførelse)
- Allokert sitt eget minne
- Det mest fundamentale konseptet i et moderne OS
- OSet lager, håndterer og terminerer prosesser
- Ready, Running, Blocked

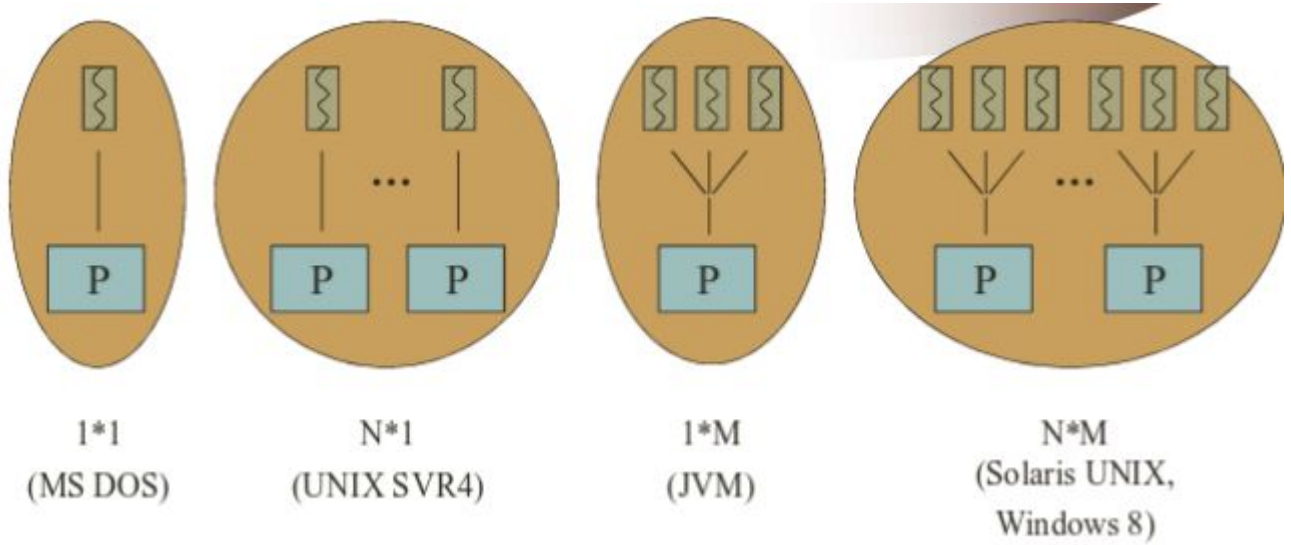
Tråder

- En tråd er en komponent av en prosess: flere samtidige tråder
- **Prosess:** ressurseierskap, **tråd:** programutførelse
- Tråder har sin egen kontekst, stakk og utførelsessti
- Deler minnet til prosessen

- Tråder får sin egen kontekst og stakk



- Enkelte kombinasjoner av prosess:tråd-forhold



Fordeler med tråder

- En prosess kan ha forskjellige utførelsesstier som det er naturlig å kjøre samtidig (eks: servitriser som jobber på restaurant)
- Vi kan dele variabler mellom disse utførelsesstiene (eks: servitrisene er nødt til å fordele restaurantens kunder)
- Det er mer effektivt å bytte trådkontekster enn prosesskontekster
- ***Men****:* Vi er nødt til å synkronisere mellom tråder, det er ikke alltid så lett

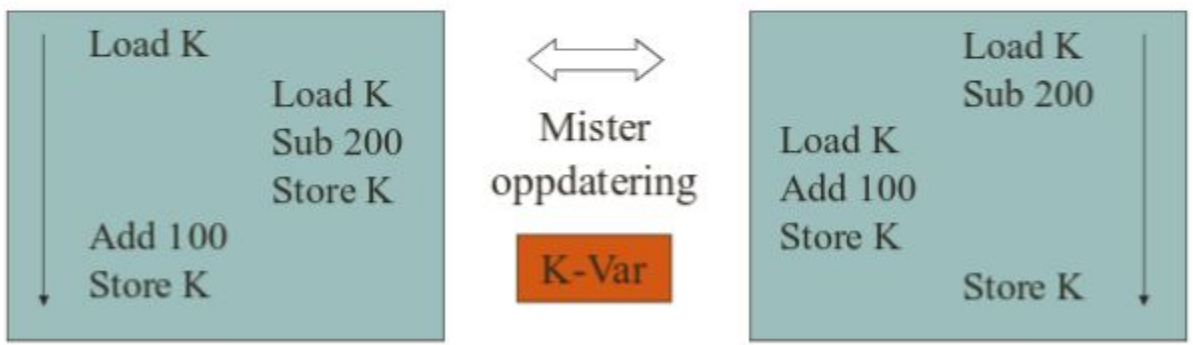
Synkronisering

- Grunnleggende krav for samtidighet er mekanismer for gjensidig utelukkelse (**mutual exclusion**)
- Samtidig aksess av delte ressurser kan føre til uventet/feilaktig oppførsel
- **Race condition**: to eller flere tråder endrer delt data samtidig, utfallet avhenger av timing (eks. neste slide)
- Vi beskytter derfor slike kritiske seksjoner i programmer med synkroniseringsmekanismer

Illustrerende problem: Bankoperasjoner

Kreditoperasjon: K := K + 100 — Load K; Add 100; Store K

Debetoperasjon: K := K - 200 — Load K; Sub 200; Store K



Synkroniseringsmekanismer

- Mutex: flagg som fungerer som portvakt for kritiske seksjoner
- Semafor: heltallsverdi som brukes for signalisering, kan aksesseres kun via *atomiske* operasjoner: initialize, decrement, increment
- Monitor: det kan være vanskelig å se effekten av mange semaforer spredt utover et program, monitor kontrollerer f.eks. objektaksess (eks: Javas *synchronized,* mer info etterpå)

Spørsmål?

Agenda

- Generelt om øvingsopplegget
- Introduksjon til tråder og synkronisering
- Presentasjon av praktisk øving P1

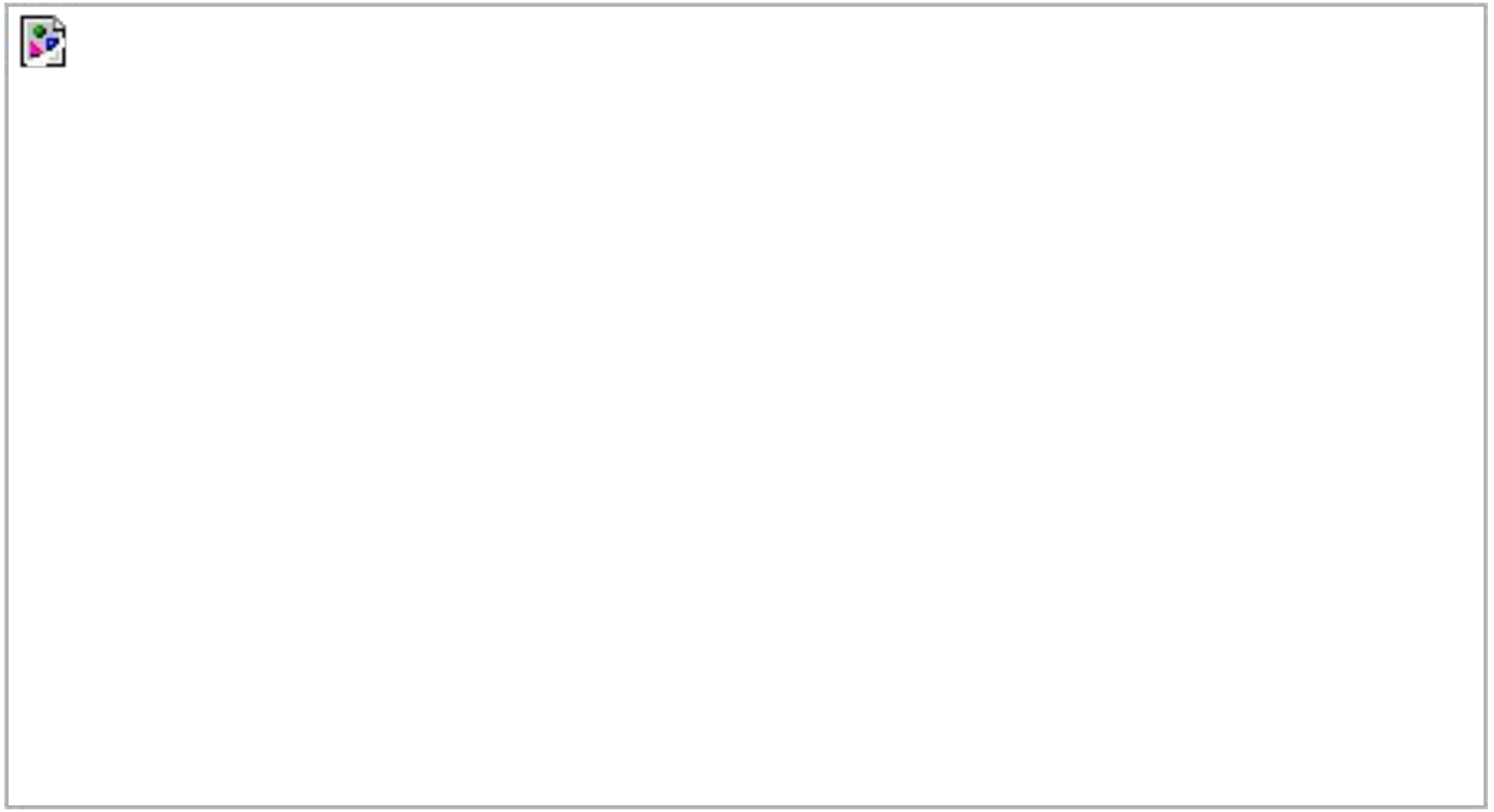
P1 – Sushibar

- **Obligatorisk**
- Form for Producer/Consumer-problem
- Fullfør klassene
- Print ut relevant informasjon, og statistikk på slutten.

Sushibar - producer/consumer

- SushiBar består av en klokke, en dør, venterom, servitriser og kunder
- Waitress er consumer, som henter kunder fra WaitingArea
- Door er producer, som lager nye kunder og flytter de inn i WaitingArea så lenge restauranten er åpen
- WaitingArea er en felles ressurs, delt av Door og Waitress, her er synkronisering viktig
- Nærmere informasjon finnes i dokumentet «Praktisk Øving»

WaitingArea



Tråder i Java

```
public class ThreadObject
    extends Thread {

    @override
    public void run(){

    }

}
```

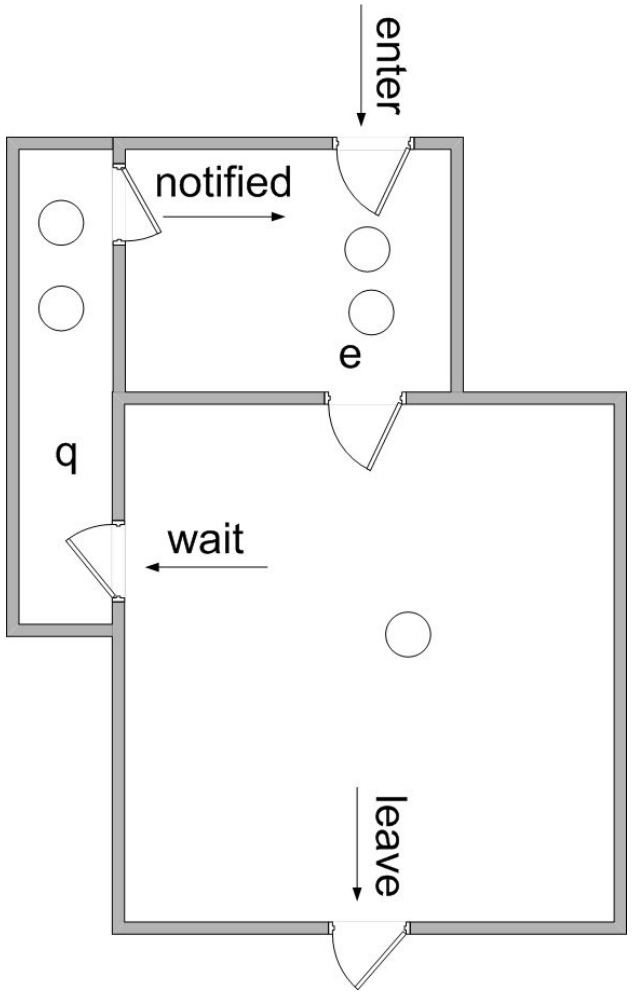
```
public class RunnableObject
    implements Runnable {

    @override
    public void run(){

    }

}
```

Starte en tråd

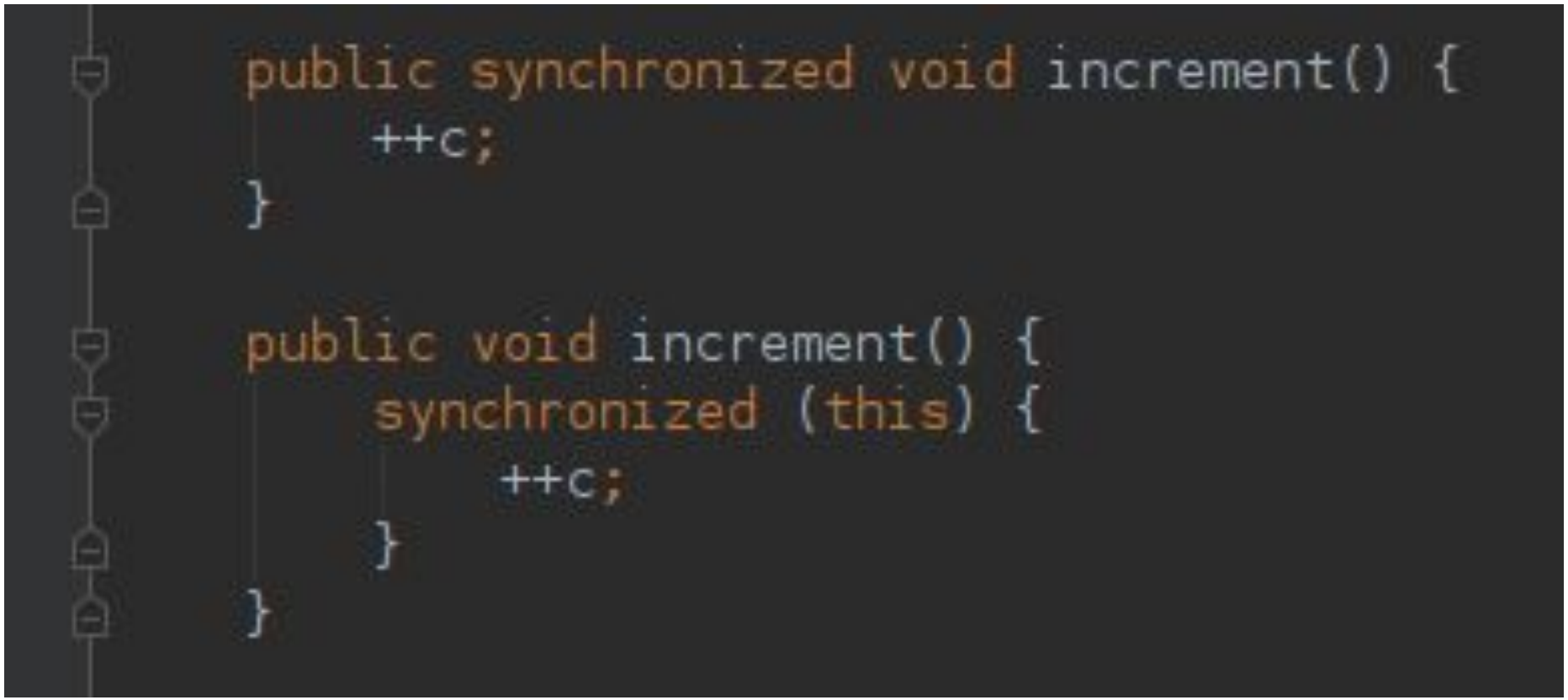
```
public static void main(String[] args) {

    new ThreadObject().start();

    new Thread(new RunnableObject()).start();

}
```

Synkronisering

- **Synchronized** nøkkelord i Java, dette er en monitor
- Alle objekter har en egen monitor
- Kun én tråd kan gå inn i monitor om gangen



Java-monitor

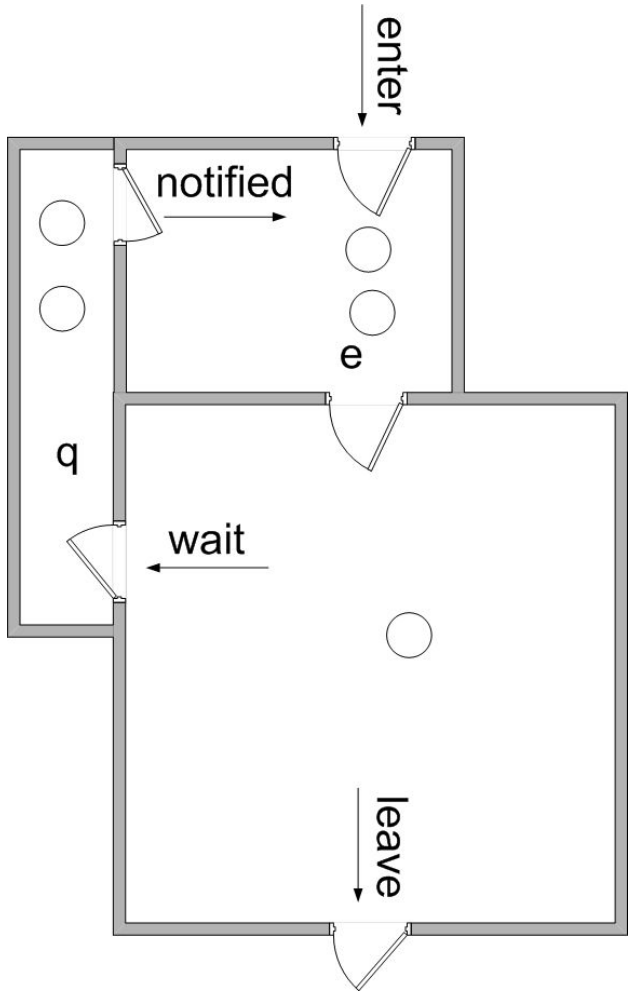

public synchronized void increment() {
++c;
}
public void increment() {
synchronized (this) {
++c;
}
}

wait/notify/notifyAll

- Erstatter polling

- **wait**() - får tråden til å gi opp låsen på monitoren, slik at en annen tråd kan ta over (se illustrasjon)
- **notify**() - vekker én tråd som tidligere har kalt wait()
- **notifyAll**() - vekker alle tråder som har kalt wait() -- førstemann til mølla

- Opp til deg å finne ut når disse er nyttige i øvingen
- *Hint: Hva skjer dersom WaitingArea er full når Door produserer en ny Customer, eller tom når en Waitress forsøker å hente en Customer?*



Tips

- Begynn med å få et overblikk over alle de utdelte klassene, hva er formålet til hver enkelt av dem? Hvordan er de ment til å fungere sammen?
- Ikke dryss nøkkelordet *synchronized* over alt uten å virkelig forstå hvorfor/hvordan det brukes
- Lek med *synchronized* (monitor) i Java - forsøk å få en intuisjon for hvilke feil som kan oppstå dersom tråder ikke synkroniserer seg i mellom
- Det kreves ikke veldig mye kode, så for å gjenta: prøv å forstå helheten før du begynner med å implementere


Spørsmål?