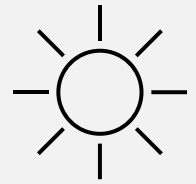
GRUPPE 11 IN1010 – UKE 9

Sivert Fjeldstad Madsen

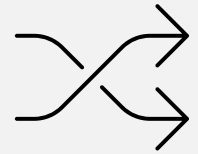
I DAG

- Parallellprogrammering
- Tråder
- Låser/Monitorer
- Conditions
- Livekoding
- Oppgaver



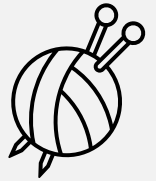
PARALLELLPROGRAMMERING

- Moderne datamaskiner har mer enn én prosessorkjerne
- Vi kan kode programmer som utnytter flere kjerner samtidig
- Dette gjør at vi kan effektivisere tunge oppgaver
  - Sortere beholdere
  - Finne primtall
  - Hente informasjon fra internett
- Til dette bruker vi **tråder**



TRÅDER

- **Thread** er et innebygd objekt i Java som kan starte en ny tråd
- Som parameter trenger en tråd et objekt som implementerer interfacet **Runnable**
- Deretter kan man kalle på metoden **start()** på tråd-objektet for å sette i gang en ny separat tråd
- Det er koden i arbeider-objektet som utføres i den nye tråden



ARBEIDERKLASSE

- Arbeider-klassen kan hete akkurat hva vi vil
  - …men gjerne noe beskrivende
- Klassen må implementere **Runnable**, som krever at vi skriver metoden **public void run()**
  - Det er i denne metoden vi skriver koden vi vil at arbeideren skal utføre

```
public Arbeider implements Runnable {
    @Override
    public void run() { … }

}
```

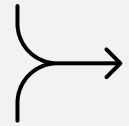
OPPRETTE OG SETTE I GANG TRÅDER

- Ofte vil vi opprette mer enn én tråd
- Da kan vi gjøre det i f.eks en for-løkke:

```
Thread[] traader = new Thread[5];
for (int i = 0; i < 5; i++) {
    traader[i] = new Thread(new Arbeider());
    traader[i].start();
}
```

VENTE PÅ TRÅDER

- Kanskje vil vi vente med å fortsette main-tråden vår til alle trådene er ferdige



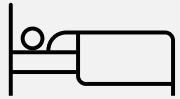
- Det kan vi gjøre med (f.eks) **join()**

```
for (int i = 0; i < 5; i++) {
    try {
        traader[i].join();
    } catch (InterruptedException e) {
        // Haandter exception
}
```

}

SOVENDE TRÅDER

- En tråd kan «sove» et gitt antall millisekunder
- Dette kan av og til være lurt f.eks om vi ønsker å se ting skje i et litt langsommere tempo

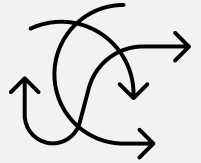


```
try {
    traad.sleep(1000);
} catch (InterruptedException e) {
    // Haandter exception
}
```

TRÅDER I SAMME MINNEOMRÅDE

- La oss si at vi har et program som skal telle ett og ett tall opp til en million
- For å gjøre programmet mer effektivt oppretter vi 10 tråder som hver øker telleren med ett og ett tall 100000 ganger hver
- Da kan vi parallelt telle til en million!
- Er det noe spesielt vi må tenke på i dette tilfellet?
  - Se filen ParallellTil1000000.java

LÅSER OG MONITORER

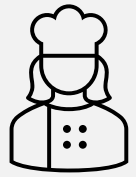


- Når flere tråder jobber i samme minneområde kan vi få **synkroniseringsproblemer**
- Flere tråder kan prøve å lese og skrive til samme minneområde samtidig, og da får vi ofte feil svar
- Det finnes flere måter å løse dette på, men i IN1010 jobber vi hovedsakelig med **låser**
- Som oftest oppretter vi en egen klasse for hver data vi trenger å låse for synkronisering, og denne klassen kalles da en **monitor**

TRY / FINALLY

- Når vi bruker låser er vi nødt til å forsikre oss om at låsen *alltid* blir låst opp igjen etter bruk
  - Dette gjør at programmet vårt ikke nødvendigvis kræsjer selv om én tråd kræsjer
- Vi putter koden vi vil skal utføres mens en tråd har låsen i en **try**-blokk, låser deretter opp låsen igjen i en **finally**-blokk
- Det som står i **finally**-blokken blir *alltid* utført – selv hvis tråden kræsjer

CONDITIONS



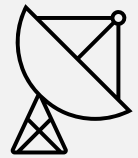
- Av og til har vi flere *ulike* arbeidere som jobber i samme minneområde
- For eksempel kan man ha:
  - Én type arbeider som «produserer» data og legger dem i en beholder
  - Én type arbeider som henter ut data fra beholderen når den første arbeidertypen har lagt noe inn
- Her kan det skje to ting:
  - Beholderen er full, og den første arbeideren må vente til noen har hentet data ut
  - Beholderen er tom, og den andre arbeideren må vente til det kommer data inn
- Hvis arbeiderene bare sjekker hele tiden bruker veldig mye overflødig prosessortid!

CONDITIONS - FORTSETTELSE

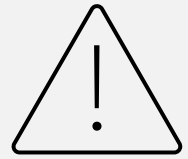
- Med conditions kan vi la trådene vente *passivt*
  - Vi lar dem «sove» til de får beskjed om å våkne
- En condition opprettes via en lås:

```
Lock laas = new ReentrantLock();
Condition cond = laas.newCondition();
```

- En tråd kan vente på at noe skal skje ved å kalle på **await()**
  - Dette gjør faktisk at tråden midlertidig låser opp låsen den holder
- En annen tråd kan gi beskjed (sende et signal) til trådene som venter med **signal()** eller **signalAll()**



TRÅDER OG EXCEPTIONS



- Når vi jobber med tråder kan det være litt tricky å huske akkurat hvor vi må håndtere exceptions
- Her en liten oversikt:
  - **join()** og **sleep()** må skje i en **try / catch**-blokk hvor vi fanger **InterruptedException**
  - Når vi låser en lås må vi alltid huske å bruke **try / finally** for å låse den opp igjen
  - **await()** «kaster» **InterruptedException**, men siden vi som oftest har dette inne i en **try / finally** er det vanlig at vi «kaster» denne exceptionen videre ut og håndterer den et annet sted

OPPSUMMERING

- Tråder er gøy!
- Det kan være mye å holde styr på, så prøv få dere oversikt
- Som vanlig: kod masse! ☺

- Ukas tips:
  - Debugger i VSCode
  - Et svært kraftig verktøy som kan gjøre det enklere å finne ut hvor feil ligger