IN1010 - OBJEKTORIENTERT PROGRAMMERING - GRUPPE 5

VELKOMMEN TIL NIENDE SEMINARTIME

Timen starter 10:15

DAGSORDEN

- Repetisjon
- Live-programmering
- Spørsmålsrunde



DAGSORDEN

- Repetisjon
- Live-programmering
- Spørsmålsrunde



REPETISJON

- Tråder
- Runnable-interfacet
- Thread-klassen
- Felles ressurser
- Låser

EKSEMPLER OG OPPGAVER

- Man kan ta i bruk tråder for å kunne utføre arbeid i parallell
- Målet er å oppnå høyere effektivitet



VS



REPETISJON

- Tråder
- Runnable-interfacet
- Thread-klassen
- Felles ressurser
- Låser

EKSEMPLER OG OPPGAVER

- Klassen til objekter som skal jobbe i parallell må implementere *Runnable*-interfacet
- Da må man @Override'e *run*-metoden
- Alt arbeidet som skal utføres i parallell må stå i *run*-metoden



```
public class Stemmeteller implements Runnable
{
    public void run()
    {
        // arbeidet som skal utfores i parallell
    }
}
```

REPETISJON

- Tråder
- Runnable-interfacet
- **Thread-klassen**
- Felles ressurser
- Låser

EKSEMPLER OG OPPGAVER

- Man bruker *Thread*-klassen til å opprette nye tråder
- Konstruktøren tar inn et *Runnable*-objekt
- For å starte starten bruker man *start*-metoden. Den kaller *run*-metoden i *Runnable*-objektet

```
public class Hovedprogram
{
    public static void main(String[] args)
    {
        Stemmeteller stemmeteller = new Stemmeteller();
        Thread traad = new Thread(stemmeteller);
        traad.start();
    }
}
```

REPETISJON

- Tråder
- Runnable-interfacet
- **Thread-klassen**
- Felles ressurser
- Låser

EKSEMPLER OG OPPGAVER

- *Thread*-klassen tilbyr også andre metoder som er verdt å nevne:
  - *sleep*-metoden tar inn antall millisekunder tråden skal vente
  - *interrupt*-metoden som signaliserer at tråden er avbrutt og bør stoppe
  - *join*-metoden brukes til å vente på at en tråd skal bli ferdig før tråden metoden kalles fra fortsetter med sin kode

REPETISJON

- Tråder
- Runnable-interfacet
- Thread-klassen
- **Felles ressurser**
- Låser

EKSEMPLER OG OPPGAVER

- Tråder kan og vil ofte ha tilgang til felles ressurser
- Metodene som opererer på disse kalles kritiske regioner
- At flere tråder opererer på felles ressurser kan føre til kappløps-tilstander
- Kappløp handler om at utfallet av arbeidet trådene utfører avhenger av rekkefølgen arbeidet blir gjort i

REPETISJON

- Tråder
- Runnable-interfacet
- Thread-klassen
- Felles ressurser
- Låser

EKSEMPLER OG OPPGAVER

- Man bruker låser til å beskytte kritiske regioner
- Vi bruker en *ReentrantLock*-lås til å sørge for at kun én tråd får operere på felles ressurser om gangen
- Husk å legge opplåsingen i en *finally*-blokk!

```
Lock laas = new ReentrantLock();

laas.lock();
try
{
    // kritisk region
}
finally { laas.unlock(); }
```

LIVE-KODING

SKRIV KODEN MED MEG 👩‍💻 👩‍💻 👩‍💻

SPØRSMÅLSRUNDE

VI SEES

HADE 👋