IN1010 Gruppetime – Uke 10

Gruppe 7

Dagens tema: Tråder 2 - Electric Boogaloo

Repetisjon

Tråder

- Hvordan oppnå parallellitet i Java? TRÅDER!!!
- En tråd er en sekvens med instruksjoner i et program
  - Noe kjent: main-metoden er en tråd som kjøres
- Vi kan kjøre flere tråder samtidig for å oppnå parallellitet
- Tråder kan vente, sove og avbrytes
- IN1010: Trådene skal ikke kommunisere med hverandre (men det er en mulighet)

Tråder i Java

- Tråder utfører *oppgaver*, og vi kan definere hva disse oppgavene som skal være
  - Eget interface: *Runnable*
  - Runnable tilbyr én metode, *run()*
- Vi kan lage egne klasser som implementerer runnable, og definere hva trådene skal gjøre i run() metoden
- For å lage en tråd bruker vi klassen *Thread*, og gir objektet en instans av oppgaveklassen vi lagde
- Når vi kaller på metoden *start()* på tråden, så vil den kjøre run() metoden

Hva skjer hvis to tråder prøver å oppdatere en verdi samtidig?

Synkronisering med låser

- For å forhindre slike feil, kan vi sørge for at kun én tråd prøver å oppdatere en felles verdi av gangen
  - Andre tråder må vente på sin tur
  - Det kalles en *kritisk region* når kun én tråd kan aksessere en verdi om gangen
- Vi bruker objektet ReentrantLock, som implementerer grensesnittet Lock
  - Bl.a to metoder, lock() og unlock()

Nytt

Barrierer

Barrierer

- En barriere er en synkroniseringsmekanisme som gjør at tråder kan møtes på et bestemt punkt i programmet
- Hvis vi tråder kjører i parallell, så vil vi ofte gå tilbake til sekvensiell kjøring etter trådene er ferdige
- Barrierer kan sikre at én eller flere tråder ikke går videre før én eller flere andre tråder har nådd barrieren.
- De gjør at vi kan kontrollere rekkefølgen av operasjoner på en trygg måte.
- To typer barrierer (i IN1010):
  - .join()
  - CountDownLatch

.join()

- .join() er en funksjon som en tråd kan kalle på
- Funksjonen gjør at tråden som funksjonen blir kalt *i* vil vente på at tråden som blir kalt *på* er ferdig
  - For eksempel: Hvis vi skriver *traad.join()* i main, så vil main-tråden vente på at *traad* kjører ferdig før den fortsetter å kjøre
  - Uten en barriere så kunne main-metoden potensielt prøvd å gjøre operasjoner på en verdi produsert av *traad* før verdien er blitt produsert

```java
public class TacoPreparation {
    public static void main(String[] args) {
        Thread vegetableThread = new Thread(new ChopVegetablesTask());
        Thread meatThread = new Thread(new CookMeatTask());
        Thread tortillaThread = new Thread(new HeatTortillasTask());
        vegetableThread.start();
        meatThread.start();
        tortillaThread.start();

        try {
            vegetableThread.join();
            meatThread.join();
            tortillaThread.join();
            System.out.println("All tasks are completed! Serve the tacos!");
        } catch (InterruptedException e) {
            System.err.println("Main thread was interrupted. Dinner is cancelled!");
            System.exit(1);
        }
    }}
```

CountDownLatch

- CountDownLatch er et objekt i Java
- CountDownLatch tar inn et heltall som parameter, som bestemmer hvor mange ganger latchen må telles ned før ventende tråder kan passere den.
- Hvis vi har et CountDownLatch-objekt *latch*:
  - En tråd kan kalle på latch.countDown() for å telle ned én gang
  - En tråd som kaller på latch.await() må vente til latch åpnes før den kan fortsette å kjøre.
  - latch åpnes når heltallet når 0
- Krever at vi vet hvor mange tråder vi har, eller vil vente på

```java
public class TacoPreparation {
    public static void main(String[] args) {
        CountDownLatch latch = new CountDownLatch(2);


        Thread vegetableThread = new Thread(new ChopVegetablesTask(latch));
        Thread meatThread = new Thread(new CookMeatTask(latch));
        Thread tortillaThread = new Thread(new HeatTortillasTask(latch));


        vegetableThread.start();
        meatThread.start();
        latch.await();
        tortillaThread.start();
    }
}
```

Livekode

Den verste sang-teksten ever