Tråder del 1

uke 9

Plan for i dag

- Gjennomgang av ukens tema
- Live koding
- Quiz (hvis tid)
- Oppgavejobbing

Hva er en tråd

- En tråd er en sekvens av instruksjoner i et program
- Hittil har vi skrevet vanlig, ikke-parallelle programmer. Disse har hatt én sekvens av instruksjoner, altså én tråd.
- Hvis vi ønsker at to eller flere oppgaver (sett med instrukser) skal kjøre samitidig, kan vi lage flere tråder.
- Ulike tråder kan dele på objekter (minne), og "snakke" med hverandre.
- I IN1010 skal trådene ikke kommunisere direkte, kun bruke delte objekter (som kalles monitorer)

Hvorfor trenger vi tråder?

- Gjør programmet mer effektivt
- Dersom vi har mye data, er det mer effektivt å ha flere tråder som jobber samtidig
- Eks en fullstappet restaurant trenger flere enn en servitør

Thread = Worker, Runnable = task

- Vi kan tenke på en tråd som en arbeider - noe som kan utføre et sett med instrukser, en oppgave. Arbeiderne er objekter av klassen Thread.
- Oppgavene som trådene kan utføre er objekter av grensesnittet Runnable.

Runnable

- Vi beskriver oppgaver som kan utføres ved objekter av grensesnittet Runnable:
- Live koding: program som sier hei flere ganger

```
interface Runnable {
    void run();
}
```

Thread

- Vi lager nye arbeidere ved å lage et objekt av klassen Thread. Vi kan så gi tråden en oppgave den kan utføre, og så be den starte.
- Vi starter opp tråder med .start()
- En tråd kan sove ved å si .sleep(), men dette kan kaste InterruptedException
- Vente på andre traader: .join()
- Forts live kode

Synkronisering (vis MyTask2)

- Dersom to tråder har tilgang på delt data, kan vi få uheldige resultater dersom vi ikke passer på.
- Problemet er at begge trådene forsøker å oppdatere samme verdi samtidig
- Dette kalles en race condition, og er (kanskje) den største kilden til feil i parallelle programmer.

Løsning: Kritisk region

- Vi løser problemet ved at kun en tråd kan aksessere et delt objekt til en gitt tid
- Vi lager en såkalt mutex, en kritisk region hvor kun en tråd er om gangen.
- Andre tråder må stoppe og vente på sin tur.

Locks

- Vi lager kritiske regioner med låser. Tenk at du har dører på hver side av koden du vil beskytte.
- Hvis trådene låser døren når de går inn, må de andre trådene vente til tråden som låste låser opp igjen.
- Vi bruker java.util.concurrent.locks.Lock, et grensesnitt som spesifiserer (blant annet) to metoder: lock() og unlock().
- Den faktiske klassen vi lager låsobjekter av heter java.util.concurrent.locks.ReentrantLock.
- forts live koding: endre MyTask2 til å ta med lock

Locks forts.

- Nå kan kun en tråd telle opp om gangen
- Merk! try{...} finally{lock.unlock();} sørger for at låsen blir låst opp.

Live koding Bombespill

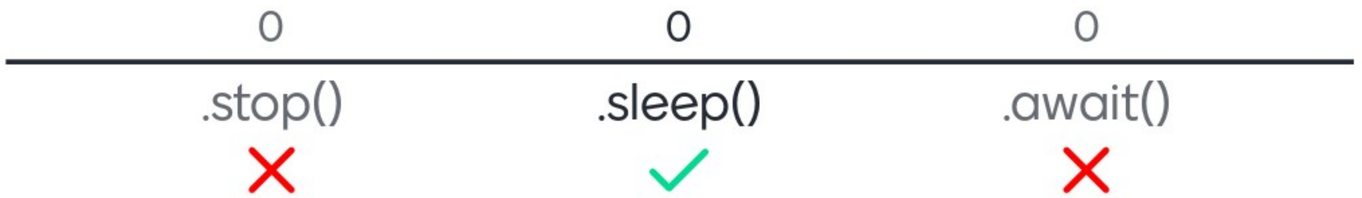
Vi lager et spill som armerer en bombe med en spesiell kode. Bomben aktiveres, og vil sprenge etter 5 sekunder med mindre spilleren taster inn koden korrekt i tide.

Tråder gjør at programmet alltid kjører raskere

Mentimeter

En tråd kan sove ved å kalle på ...

Mentimeter



Ved å kalle på Thread.sleep() kan det kastes...

Mentimeter

Brukes for å sørge for at kun en tråd bruker ressursen om gangen

Mentimeter

Finally sørger for ..

Mentimeter