Tråder del 1

in1010 – gruppe 2

25 mars 2025

this week>

- Hva er tråder?
- Synkronisering
- Livekoding
- Oppgaver

Hva er tråder?

- En tråd er en **sekvens av instruksjoner**.
- Sekvensielle programmer kjører i **bestemt rekkefølge**
- Hittil har vi bare hatt én tråd: public static void main(String[] args)
- Dersom vi ønsker å kjøre to eller flere sett med **instruksjoner samtidig**, kan vi lage flere tråder.
- Å kjøre flere tråder sammen for å dele på arbeidsoppgavene heter **parallellitet**.
    - Får gjort mer på samme tid

Hvorfor tråder?

- Dele på arbeidsoppgavene slik at programmet kjører raskere
- Bedre ytelse og mer effektivt

Tråder kan

- Tråder kan
  - starte ved .start()
  - sove med .sleep()
  - vente med .wait() eller på andre tråder med .join()
  - avbrytes med .interrupt()
- De kan dele felles data (minnet)
- De kan også snakke med hverandre(ikke en del av in1010), vi skal kun bruke delte objekter

Interface Runnable

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

- **Oppgaven** til tråden blir laget ved å implementere **grensesnittet Runnable**
- Må implementere **run()** metoden til Runnable.
- Instruksjonene/oppgavene som vi vil at tråden skal utføres skal ligge i run()

Eksempel: MyTask

- Et eksempel på et tråd-program

MyTask.java > MyTask

```java
// Eksempel 1
public class MyTask implements Runnable {
    /* Say hello, but think for a while between each time. */
    public void run() {
        for (int i = 1; i < 5000; i++)
        if (i % 1000 == 0)
        System.out.printf(format:"Hello for the %d-th time!\n", i / 1000);
    }

    Run | Debug
    public static void main(String[] args) {
        Runnable task = new MyTask();
        Thread thread1 = new Thread(task);
        Thread thread2 = new Thread(task);
        thread1.start(); thread2.start(); // Will make a call to task.run()
    }
}
```

Synkronisering

- Når tråder deler på data (f.eks instansvariabler) kan det oppstå **problemer** dersom de gjør endringer **på samme data samtidig.**
- Race condition
- Ofte feil i parallel-programmering
- Må bruke **låser**

Låser



- Løser problemet ved å bare la **én tråd aksessere** delte objektet om gangen.
  - Lager en kritisk region / monitor
- De andre trådene **må vente** på sin tur
- Bruker objekter av **ReentrantLock**, må importere java.util.concurrent.locks.Lock

Eksempel ved bruk av lås: eksempeLass

- Lager en lås slik at bare en tråd kan endre på sharedCounter
- Merk at vi låser alltid opp låsen i finally for å unngå deadlocks try{...}finally{ **lock.unlock()** }

eksempeLaas.java > ...

```java
// Eksempel 2
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class eksempeLaas implements Runnable {
    private final Lock lock = new ReentrantLock();
    private final int MAX_COUNT = 10000; private int sharedCounter = 0;

    public void run() {
        System.out.println("Starting! Shared counter = " + getSharedCounter());
        for (int i = 0; i < MAX_COUNT; i++) {
            lock.lock();
            try {
                sharedCounter = sharedCounter + 1;
            } finally {
                lock.unlock();
            }
        }
        System.out.println("Done! Shared counter = " + getSharedCounter());
    }

    private int getSharedCounter() {
        lock.lock();
        try {
            return sharedCounter;
        } finally {
            lock.unlock();
        }
    }
    Run | Debug
    public static void main(String[] args) {
        Runnable task = new eksempeLaas();
        Thread thread1 = new Thread(task);
        Thread thread2 = new Thread(task);
        thread1.start(); thread2.start(); // Will make a call to task.run()
```

Livekode: 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.

Seminaroppgaver