IN1010

UKE 11 - GRUPPE 3

MIAU@IFI.UIO.NO

Plan for dagen

- Parallellitet
- Tråder
- Trådsikkerhet
  - Lock
  - Monitor
- Barrierer
  - join()
  - CountDownLatch
- Conditions

Parallellitet

- Sekvensiell programmering
  - Koden kjøres fra topp til bunn

- Parallellitet i programmer
  - Lar oss utføre flere oppgaver samtidig
  - Oppnås ved å bruke tråder

Tråder

- En sekvens med instruksjoner i et program
- Kan kjøre flere tråder samtidig, som dermed utfører kode samtidig
- Kan vente, sove og avbrytes

- Runnable-interfacet
  - Implementere run()-metoden → Hva skal tråden gjøre?
- Lage en tråd vha. Thread-klassen
- Starte tråden ved å kalle på start()
  - Kjører run()

Tråder

```java
class Tråd implements Runnable {

    @Override
    public void run() {
        System.out.println(x: "Tråden har startet!");
    }
}
```

```java
Run | Debug
public static void main(String[] args) {
    Thread tråden = new Thread(new Tråd());
    tråden.start();
}
```

En utfordring med tråder

```java
class Tråd implements Runnable {
    static int teller = 0;

    @Override
    public void run() {
        teller++;
        System.out.println("Tråd nr." + teller + " kjører");
    }
}
```

```java
Run | Debug
public static void main(String[] args) {
    Thread tråd1 = new Thread(new Tråd());
    Thread tråd2 = new Thread(new Tråd());
    tråd1.start();
    tråd2.start();
}
```

```
Tråd nr.1 kjører

Tråd nr.2 kjører
```

```
Tråd nr.2 kjører

Tråd nr.2 kjører
```

Lock

- En lås sørger for at det kun er en tråd som har tilgang til variabler om gangen

```java
import java.util.concurrent.locks.ReentrantLock;
import java.util.concurrent.locks.Lock;

class Tråd implements Runnable {
    Lock lås = new ReentrantLock();
    int teller = 0;

    @Override
    public void run() {
        lås.lock();
        try {
            teller++;
            System.out.println("Tråd nr." + teller + " kjører");
        } finally {
            lås.unlock();
        }
    }
}
```

```java
Run | Debug
public static void main(String[] args) {
    Tråd tråd = new Tråd();
    Thread tråd1 = new Thread(tråd);
    Thread tråd2 = new Thread(tråd);
    tråd1.start();
    tråd2.start();
}
```

Monitor

- En klasse som håndterer synkroniseringen av tråder
  - Altså her man implementerer bla. Lock
- Tar kompleksiteten ut fra tråd-klassen
- Lagrer delt informasjon utenfor tråden

```java
class Tråd implements Runnable {
    Monitor monitor;

    Tråd(Monitor monitor) {
        this.monitor = monitor;
    }

    @Override
    public void run() {
        monitor.økTeller();
    }
}
```

```java
import java.util.concurrent.locks.ReentrantLock;
import java.util.concurrent.locks.Lock;

class Monitor {
    int teller = 0;
    Lock lås = new ReentrantLock();

    public void økTeller() {
        lås.lock();
        try {
            teller++;
            System.out.println("Tråd nr." + teller + " kjører");
        } finally {
            lås.unlock();
        }
    }
}
```

```java
public static void main(String[] args) {
    Monitor monitor = new Monitor();

    for (int i = 0; i < 25; i++) {
        Tråd tråd = new Tråd(monitor);
        Thread thread = new Thread(tråd);
        thread.start();
    }
}
```

Barrierer

- En barrierer hjelper oss med å sørge for at alle tråder møtes på et bestemt sted i et program
  - Vi vil ofte gå tilbake til sekvensiell kjøring når trådene har gjort sitt
- Kan også sørge for at en bestemt tråd ikke går videre før en annen tråd er ferdig
- I IN1010:
  - join()
  - CountDownLatch

.join()

Gjør at tråden .join() blir kalt i, vil vente på selve tråden som kaller på .join(), før den går videre

```
Run | Debug
public static void main(String[] args)

    Monitor monitor = new Monitor();
    Tråd tråd = new Tråd(monitor);

    Thread tråd1 = new Thread(tråd);
    Thread tråd2 = new Thread(tråd);

    tråd1.start();
    tråd1.join();

    System.out.println(x: "Tråd 1 er ferdig");

    tråd2.start();
    tråd2.join();

    System.out.println(x: "Tråd 2 er ferdig");
}
```

```
Run | Debug
public static void main(String[] args) throws InterruptedException {

    Monitor monitor = new Monitor();
    Tråd tråd = new Tråd(monitor);
    ArrayList<Thread> liste = new ArrayList<>();

    for (int i = 0; i < 100; i++) {
        Thread nyTråd = new Thread(tråd);
        liste.add(nyTråd);
    }

    for (Thread t : liste) {
        t.start();
    }

    for (Thread t : liste) {
        t.join();
    }

    // kode man vil kjøre etter ALLE trådene er ferdig

}
```

CountDownLatch

- Tar inn et heltall som parameter
  - Tilsier hvor mange ganger latchen skal telle ned, før man kan passere den

- Om vi har et CountDownLatch-objekt *latch:*
  - Om en tråd kaller latch.countDown(), teller den ned en gang
  - En tråd som kaller latch.await(), må vente til latch åpnes før den kan fortsette å kjøre
  - latch åpnes når tallet har blitt 0

- Krever at vi vet hvor mange tråder vi skal håndtere

```java
class Tråd implements Runnable {

    Monitor monitor;
    CountDownLatch latch;

    Tråd(Monitor monitor, CountDownLatch latch) {
        this.monitor = monitor;
        this.latch = latch;
    }

    @Override
    public void run() {
        try {
            monitor.økTeller();
            latch.countDown();
        } catch (InterruptedException e) {
            System.err.println();
        }
    }
}
```

```java
Run | Debug
public static void main(String[] args) {

    CountDownLatch latch = new CountDownLatch(count: 2);
    Monitor monitor = new Monitor();

    Thread tråd1 = new Thread(new Tråd(monitor, latch));
    Thread tråd2 = new Thread(new Tråd(monitor, latch));
    Thread tråd3 = new Thread(new Tråd(monitor, latch));

    tråd1.start();
    tråd2.start();

    try {
        latch.await();
    } catch (InterruptedException e) {
        System.err.println(e);
    }

    tråd3.start();
}
```

Condition

- En Condition hører til en lås
- Opprettes ved å bruke .newCondition() på låsen din

- En Condition kan:
  - condition.await() → venter til condition er oppfylt
  - condition.signal() → en tråd som venter på condition signaliseres
  - condition.signalAll() → alle trådene som venter på condition signaliseres

```java
class taUt implements Runnable {
    Bank bank;

    taUt(Bank bank) {
        this.bank = bank;
    }

    @Override
    public void run() {
        try {
            bank.taUt(penger: 100);
        } catch (InterruptedException e) {
            System.err.println(e);
        }
    }
}
```

```java
class settInn implements Runnable {
    Bank bank;

    settInn(Bank bank) {
        this.bank = bank;
    }

    @Override
    public void run() {
        bank.settInn(penger: 500);
    }
}
```

```java
class Bank {
    int saldo = 100;
    Lock lås = new ReentrantLock();
    Condition nokPenger = lås.newCondition();

    public void taUt(int penger) throws InterruptedException {
        lås.lock();
        try {
            while (saldo < penger) {
                nokPenger.await();
            }
            saldo -= penger;
            System.out.println("Saldo: " + this.saldo);
        } finally {
            lås.unlock();
        }
    }

    public void settInn(int penger) {
        lås.lock();
        try {
            saldo += penger;
            nokPenger.signalAll();
        } finally {
            lås.unlock();
        }
    }
}
```