IN1010

Repetisjon Uke 10
Tråder og parallellitet, del 2

(Basert på Stein Gjessings presentasjon)

28. mars, 2025

Produsent/konsument

Produsent:

- Skaper eller henter inn (produserer) data
- Prøver å levere det til monitoren (delt objekt)

Konsument

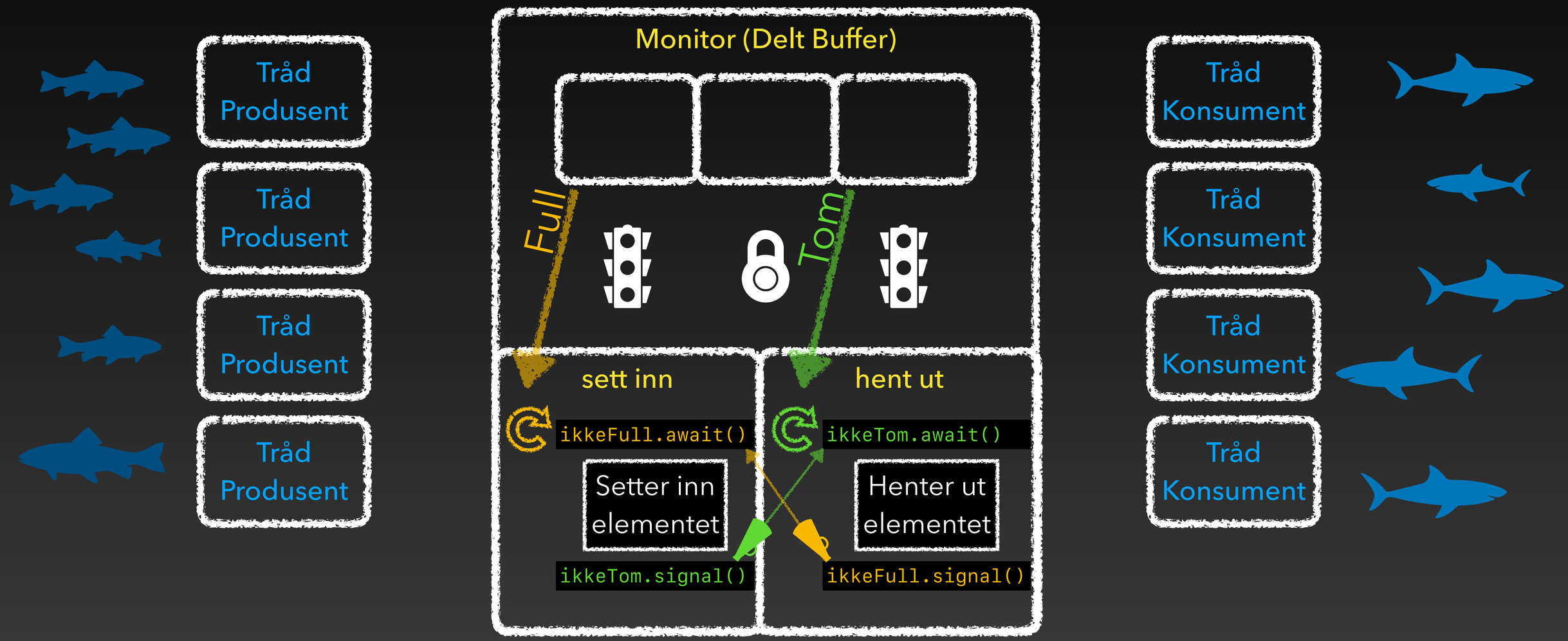
- Prøver å hente ut data fra monitoren
- Jobber med (konsumerer) dataene

Produsent/konsument

Monitor:

- En data-struktur med plass til en eller fler data-enheter
- Må være tråd-sikker, all synkronisering skjer her
- Ideen er at produsentene og konsumentene er individuelle og uavhengige tråder, som ikke trenges å synkroniseres internt

Produsent/konsument



Obligatorisk innlevering 5

Her har vi litt mer avansert monitor, og noen av trådene fungerer både som konsument (henter ut data) og produsent (fletter og setter inn igjen).

Konsumentene skal også hente ut to elementer fra beholderen, så lurt å tenke på hva som skjer når det bare er én igjen.

Barrierer

En barriere er en mekanisme som fungerer som et samlepunkt for flere tråder, hvor de venter til alle er ferdige før de fortsetter.

Det kan sammenlignes med at en gruppe er på tur, hvor man bestemmer seg for et møtepunkt der alle skal samles før man går videre på neste etappe.

Eller hvis man venter på at løpere skal gjøre seg klar før startskuddet.

Barrierer

Vi har i Java to hovedtyper av barrierer:

`CountDownLatch`

- Brukes hvis du har en hendelse du vil vente på
- Kan brukes bare en gang

`CyclicBarrier`

- Hvis du vil at programmet skal vente på et visst antall tråder
- Kan brukes flere ganger

```
class Sau implements Runnable {
    private final String navn;
    private final CountDownLatch klar;
    Sau(String navn, CountDownLatch klar) {
        this.navn = navn; this.klar = klar;
    }
    @Override
    public void run() {
        try {
            System.out.println("Sauen " + navn + " er klar for start!");
            klar.countDown();
        } catch (InterruptedException e) { … }
    }
}
class Race {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        CountDownLatch klarTilStart = new CountDownLatch(ANTALL_SAUER);
        Thread[] saueThreads = new Thread[ANTALL_SAUER];
        for (int i = 0; i < ANTALL_SAUER; i++) {
            saueThreads[i] = new Thread(new Sau(("Sau " + i), klarTilStart);
            saueThreads[i].start();
        }
        try {
            System.out.println("Sauene gjør seg klare.");
            klarTilStart.await();
            System.out.println("Alle er klare!");
            System.out.println("Løøøp!");
        } catch (InterruptedException e) { … }
    }
}
```

CountDownLatch

```
class Sau implements Runnable {
    private final String navn;
    private final CountDownLatch klar;
    Sau(String navn, CountDownLatch klar) {
        this.navn = navn; this.klar = klar;
    }
    @Override
    public void run() {
        try {
            System.out.println("Sauen " + navn + " er klar for start!");
            klar.countDown();
        } catch (InterruptedException e) { … }
    }
}
class Race {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        CountDownLatch klarTilStart = new CountDownLatch(ANTALL_SAUER);
        Thread[] saueThreads = new Thread[ANTALL_SAUER];
        for (int i = 0; i < ANTALL_SAUER; i++) {
            saueThreads[i] = new Thread(new Sau(("Sau " + i), klarTilStart);
            saueThreads[i].start();
        }
        try {
            System.out.println("Sauene gjør seg klare.");
            klarTilStart.await();
            System.out.println("Alle er klare!");
            System.out.println("Løøøp!");
        } catch (InterruptedException e) { … }
    }
}
```

Først oppretter vi en CountDownLatch med et bestemt antall

```
class Sau implements Runnable {
    private final String navn;
    private final CountDownLatch klar;
    Sau(String navn, CountDownLatch klar) {
        this.navn = navn; this.klar = klar;
    }
    @Override
    public void run() {
        try {
            System.out.println("Sauen " + navn + " er klar for start!");
            klar.countDown();
        } catch (InterruptedException e) { … }
    }
}
class Race {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        CountDownLatch klarTilStart = new CountDownLatch(ANTALL_SAUER);
        Thread[] saueThreads = new Thread[ANTALL_SAUER];
        for (int i = 0; i < ANTALL_SAUER; i++) {
            saueThreads[i] = new Thread(new Sau(("Sau " + i), klarTilStart);
            saueThreads[i].start();
        }
        try {
            System.out.println("Sauene gjør seg klare.");
            klarTilStart.await();
            System.out.println("Alle er klare!");
            System.out.println("Løøøp!");
        } catch (InterruptedException e) { … }
    }
}
```

Så oppretter vi Sau-objekter som også har en tråd hver

```
class Sau implements Runnable {
    private final String navn;
    private final CountDownLatch klar;
    Sau(String navn, CountDownLatch klar) {
        this.navn = navn; this.klar = klar;
    }
    @Override
    public void run() {
        try {
            System.out.println("Sauen " + navn + " er klar for start!");
            klar.countDown();
        } catch (InterruptedException e) { … }
    }
}
class Race {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        CountDownLatch klarTilStart = new CountDownLatch(ANTALL_SAUER);
        Thread[] saueThreads = new Thread[ANTALL_SAUER];
        for (int i = 0; i < ANTALL_SAUER; i++) {
            saueThreads[i] = new Thread(new Sau(("Sau " + i), klarTilStart);
            saueThreads[i].start();
        }
        try {
            System.out.println("Sauene gjør seg klare.");
            klarTilStart.await();
            System.out.println("Alle er klare!");
            System.out.println("Løøøp!");
        } catch (InterruptedException e) { … }
    }
}
```

Hver tråd har en referanse til CountDownLatch'en fra main

```
class Sau implements Runnable {
    private final String navn;
    private final CountDownLatch klar;
    Sau(String navn, CountDownLatch klar) {
        this.navn = navn; this.klar = klar;
    }
    @Override
    public void run() {
        try {
            System.out.println("Sauen " + navn + " er klar for start!");
            klar.countDown();
        } catch (InterruptedException e) { … }
    }
}
class Race {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        CountDownLatch klarTilStart = new CountDownLatch(ANTALL_SAUER);
        Thread[] saueThreads = new Thread[ANTALL_SAUER];
        for (int i = 0; i < ANTALL_SAUER; i++) {
            saueThreads[i] = new Thread(new Sau(("Sau " + i), klarTilStart);
            saueThreads[i].start();
        }
        try {
            System.out.println("Sauene gjør seg klare.");
            klarTilStart.await();
            System.out.println("Alle er klare!");
            System.out.println("Løøøp!");
        } catch (InterruptedException e) { … }
    }
}
```

Vi starter tråden, som så etterhvert kjører countDown()-metoden

```
class Sau implements Runnable {
    private final String navn;
    private final CountDownLatch klar;
    Sau(String navn, CountDownLatch klar) {
        this.navn = navn; this.klar = klar;
    }
    @Override
    public void run() {
        try {
            System.out.println("Sauen " + navn + " er klar for start!");
            klar.countDown();
        } catch (InterruptedException e) { … }
    }
}
class Race {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        CountDownLatch klarTilStart = new CountDownLatch(ANTALL_SAUER);
        Thread[] saueThreads = new Thread[ANTALL_SAUER];
        for (int i = 0; i < ANTALL_SAUER; i++) {
            saueThreads[i] = new Thread(new Sau(("Sau " + i), klarTilStart);
            saueThreads[i].start();
        }
        try {
            System.out.println("Sauene gjør seg klare.");
            klarTilStart.await();
            System.out.println("Alle er klare!");
            System.out.println("Løøøp!");
        } catch (InterruptedException e) { … }
    }
}
```

Den er en del av CountDownLatch, og teller ned med 1

```
class Sau implements Runnable {
    private final String navn;
    private final CountDownLatch klar;
    Sau(String navn, CountDownLatch klar) {
        this.navn = navn; this.klar = klar;
    }
    @Override
    public void run() {
        try {
            System.out.println("Sauen " + navn + " er klar for start!");
            klar.countDown();
        } catch (InterruptedException e) { … }
    }
}
class Race {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        CountDownLatch klarTilStart = new CountDownLatch(ANTALL_SAUER);
        Thread[] saueThreads = new Thread[ANTALL_SAUER];
        for (int i = 0; i < ANTALL_SAUER; i++) {
            saueThreads[i] = new Thread(new Sau(("Sau " + i), klarTilStart);
            saueThreads[i].start();
        }
        try {
            System.out.println("Sauene gjør seg klare.");
            klarTilStart.await();
            System.out.println("Alle er klare!");
            System.out.println("Løøøp!");
        } catch (InterruptedException e) { … }
    }
}
```

Mens Sau-trådene kjører, så i main-tråden kommer vi til await()

```
class Sau implements Runnable {
    private final String navn;
    private final CountDownLatch klar;
    Sau(String navn, CountDownLatch klar) {
        this.navn = navn; this.klar = klar;
    }
    @Override
    public void run() {
        try {
            System.out.println("Sauen " + navn + " er klar for start!");
            klar.countDown();
        } catch (InterruptedException e) { … }
    }
}
class Race {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        CountDownLatch klarTilStart = new CountDownLatch(ANTALL_SAUER);
        Thread[] saueThreads = new Thread[ANTALL_SAUER];
        for (int i = 0; i < ANTALL_SAUER; i++) {
            saueThreads[i] = new Thread(new Sau(("Sau " + i), klarTilStart);
            saueThreads[i].start();
        }
        try {
            System.out.println("Sauene gjør seg klare.");
            klarTilStart.await();
            System.out.println("Alle er klare!");
            System.out.println("Løøøp!");
        } catch (InterruptedException e) { … }
    }
}
```

Her venter main-tråden til countDown() har telt til 0

Barrierer

`CountDownLatch`

- Opprettes som et objekt med et bestemt tall
- Hvert kall på countDown() teller ned fra det tallet
- Er ikke avhengig av antall tråder, men hvor mange ganger countDown() kjøres.
- Når tallet har nådd 0, så fortsetter programmet etter await()-metoden til CountDownLatch-objektet

```
class Sau implements Runnable {
    private final String navn;
    private final CyclicBarrier ventepunkt;
    Sau(String navn, CyclicBarrier ventepunkt) {
        this.navn = navn; this.ventepunkt = ventepunkt;
    }
    @Override
    public void run() {
        try {
            for (int i = 1; i <= 3; i++) {
                System.out.println(navn + " venter på de andre");
                ventepunkt.await();
                System.out.println(navn + " fortsetter sammen med flokken");
                Thread.sleep(1000); // Simuler at sauen går
            }
            System.out.println(navn + " har nådd setra.");
       } catch (InterruptedException | BrokenBarrierException e) { … }
    }
}

class TilSeters {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        System.out.println("Turen til setra");
        CyclicBarrier etappepunkter = new CyclicBarrier(ANTALL_SAUER);
        for (int i = 0; i < ANTALL_SAUER; i++) {
            new Thread(new Sau(("Sau " + i), etappepunkter)).start();
        }
        // Eventuelt f.eks. .join av alle trådene
    }
}
```

CyclicBarrier

```
class Sau implements Runnable {
    private final String navn;
    private final CyclicBarrier ventepunkt;
    Sau(String navn, CyclicBarrier ventepunkt) {
        this.navn = navn; this.ventepunkt = ventepunkt;
    }
    @Override
    public void run() {
        try {
            for (int i = 1; i <= 3; i++) {
                System.out.println(navn + " venter på de andre");
                ventepunkt.await();
                System.out.println(navn + " fortsetter sammen med flokken");
                Thread.sleep(1000); // Simuler at sauen går
            }
            System.out.println(navn + " har nådd setra.");
       } catch (InterruptedException | BrokenBarrierException e) { … }
    }
}

class TilSeters {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        System.out.println("Turen til setra");
        CyclicBarrier etappepunkter = new CyclicBarrier(ANTALL_SAUER);
        for (int i = 0; i < ANTALL_SAUER; i++) {
            new Thread(new Sau(("Sau " + i), etappepunkter)).start();
        }
        // Eventuelt f.eks. .join av alle trådene
    }
}
```

Vi oppretter et objekt av CyclicBarrier med et antall

```
class Sau implements Runnable {
    private final String navn;
    private final CyclicBarrier ventepunkt;
    Sau(String navn, CyclicBarrier ventepunkt) {
        this.navn = navn; this.ventepunkt = ventepunkt;
    }
    @Override
    public void run() {
        try {
            for (int i = 1; i <= 3; i++) {
                System.out.println(navn + " venter på de andre");
                ventepunkt.await();
                System.out.println(navn + " fortsetter sammen med flokken");
                Thread.sleep(1000); // Simuler at sauen går
            }
            System.out.println(navn + " har nådd setra.");
       } catch (InterruptedException | BrokenBarrierException e) { … }
    }
}

class TilSeters {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        System.out.println("Turen til setra");
        CyclicBarrier etappepunkter = new CyclicBarrier(ANTALL_SAUER);
        for (int i = 0; i < ANTALL_SAUER; i++) {
            new Thread(new Sau(("Sau " + i), etappepunkter)).start();
        }
        // Eventuelt f.eks. .join av alle trådene
    }
}
```

Dette antallet bestemmer antall tråder som barrieren skal vente på

```
class Sau implements Runnable {
    private final String navn;
    private final CyclicBarrier ventepunkt;
    Sau(String navn, CyclicBarrier ventepunkt) {
        this.navn = navn; this.ventepunkt = ventepunkt;
    }
    @Override
    public void run() {
        try {
            for (int i = 1; i <= 3; i++) {
                System.out.println(navn + " venter på de andre");
                ventepunkt.await();
                System.out.println(navn + " fortsetter sammen med flokken");
                Thread.sleep(1000); // Simuler at sauen går
            }
            System.out.println(navn + " har nådd setra.");
       } catch (InterruptedException | BrokenBarrierException e) { … }
    }
}

class TilSeters {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        System.out.println("Turen til setra");
        CyclicBarrier etappepunkter = new CyclicBarrier(ANTALL_SAUER);
        for (int i = 0; i < ANTALL_SAUER; i++) {
            new Thread(new Sau(("Sau " + i), etappepunkter)).start();
        }
        // Eventuelt f.eks. .join av alle trådene
    }
}
```

Vi oppretter Sau-objekter med referanse til barrieren, og starter trådene

```
class Sau implements Runnable {
    private final String navn;
    private final CyclicBarrier ventepunkt;
    Sau(String navn, CyclicBarrier ventepunkt) {
        this.navn = navn; this.ventepunkt = ventepunkt;
    }
    @Override
    public void run() {
        try {
            for (int i = 1; i <= 3; i++) {
                System.out.println(navn + " venter på de andre");
                ventepunkt.await();
                System.out.println(navn + " fortsetter sammen med flokken");
                Thread.sleep(1000); // Simuler at sauen går
            }
            System.out.println(navn + " har nådd setra.");
       } catch (InterruptedException | BrokenBarrierException e) { … }
    }
}

class TilSeters {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        System.out.println("Turen til setra");
        CyclicBarrier etappepunkter = new CyclicBarrier(ANTALL_SAUER);
        for (int i = 0; i < ANTALL_SAUER; i++) {
            new Thread(new Sau(("Sau " + i), etappepunkter)).start();
        }
        // Eventuelt f.eks. .join av alle trådene
    }
}
```

På samme måte som før har hver tråd en referanse til barrieren

```
class Sau implements Runnable {
    private final String navn;
    private final CyclicBarrier ventepunkt;
    Sau(String navn, CyclicBarrier ventepunkt) {
        this.navn = navn; this.ventepunkt = ventepunkt;
    }
    @Override
    public void run() {
        try {
            for (int i = 1; i <= 3; i++) {
                System.out.println(navn + " venter på de andre");
                ventepunkt.await();
                System.out.println(navn + " fortsetter sammen med flokken");
                Thread.sleep(1000); // Simuler at sauen går
            }
            System.out.println(navn + " har nådd setra.");
       } catch (InterruptedException | BrokenBarrierException e) { … }
    }
}

class TilSeters {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        System.out.println("Turen til setra");
        CyclicBarrier etappepunkter = new CyclicBarrier(ANTALL_SAUER);
        for (int i = 0; i < ANTALL_SAUER; i++) {
            new Thread(new Sau(("Sau " + i), etappepunkter)).start();
        }
        // Eventuelt f.eks. .join av alle trådene
    }
}
```

Her så bruker vi await() inne i tråden til objektet

```
class Sau implements Runnable {
    private final String navn;
    private final CyclicBarrier ventepunkt;
    Sau(String navn, CyclicBarrier ventepunkt) {
        this.navn = navn; this.ventepunkt = ventepunkt;
    }
    @Override
    public void run() {
        try {
            for (int i = 1; i <= 3; i++) {
                System.out.println(navn + " venter på de andre");
                ventepunkt.await();
                System.out.println(navn + " fortsetter sammen med flokken");
                Thread.sleep(1000); // Simuler at sauen går
            }
            System.out.println(navn + " har nådd setra.");
       } catch (InterruptedException | BrokenBarrierException e) { … }
    }
}

class TilSeters {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        System.out.println("Turen til setra");
        CyclicBarrier etappepunkter = new CyclicBarrier(ANTALL_SAUER);
        for (int i = 0; i < ANTALL_SAUER; i++) {
            new Thread(new Sau(("Sau " + i), etappepunkter)).start();
        }
        // Eventuelt f.eks. .join av alle trådene
    }
}
```

Ventingen vil vare helt til like mange tråder som antallet har brukt await()

```
class Sau implements Runnable {
    private final String navn;
    private final CyclicBarrier ventepunkt;
    Sau(String navn, CyclicBarrier ventepunkt) {
        this.navn = navn; this.ventepunkt = ventepunkt;
    }
    @Override
    public void run() {
        try {
            for (int i = 1; i <= 3; i++) {
                System.out.println(navn + " venter på de andre");
                ventepunkt.await();
                System.out.println(navn + " fortsetter sammen med flokken");
                Thread.sleep(1000); // Simuler at sauen går
            }
            System.out.println(navn + " har nådd setra.");
       } catch (InterruptedException | BrokenBarrierException e) { … }
    }
}

class TilSeters {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        System.out.println("Turen til setra");
        CyclicBarrier etappepunkter = new CyclicBarrier(ANTALL_SAUER);
        for (int i = 0; i < ANTALL_SAUER; i++) {
            new Thread(new Sau(("Sau " + i), etappepunkter)).start();
        }
        // Eventuelt f.eks. .join av alle trådene
    }
}
```

Etter at alle trådene har ventet, så resettes barrieren og vi kan bruke den igjen

```
class Sau implements Runnable {
    private final String navn;
    private final CyclicBarrier ventepunkt;
    Sau(String navn, CyclicBarrier ventepunkt) {
        this.navn = navn; this.ventepunkt = ventepunkt;
    }
    @Override
    public void run() {
        try {
            for (int i = 1; i <= 3; i++) {
                System.out.println(navn + " venter på de andre");
                ventepunkt.await();
                System.out.println(navn + " fortsetter sammen med flokken");
                Thread.sleep(1000); // Simuler at sauen går
            }
            System.out.println(navn + " har nådd setra.");
       } catch (InterruptedException | BrokenBarrierException e) { … }
    }
}

class TilSeters {
    private static final int ANTALL_SAUER = 5;
    public static void main(String[] args) {
        System.out.println("Turen til setra");
        CyclicBarrier etappepunkter = new CyclicBarrier(ANTALL_SAUER);
        for (int i = 0; i < ANTALL_SAUER; i++) {
            new Thread(new Sau(("Sau " + i), etappepunkter)).start();
        }
        // Eventuelt f.eks. .join av alle trådene
    }
}
```

Når vi bruker CyclicBarrier så må vi også fange et unntak hvis barrieren blir brutt

Barrierer

`CyclicBarrier`

- Opprettes som et objekt med et tall for antall tråder den skal gjelde for
- Kall på await() til barriere-objektet, venter i tråden den blir kjørt
- Trådene fortsetter når antallet await() er samme som når vi opprettet barrieren
- Opprettes for et bestemt antall tråder
- Kan brukes flere ganger (cyclic)

Tråder er lett å snuble i



Tråder er lett å snuble i



Pass på at du ikke kjører trådene i sekvens

```
Thread tråd;
for (int i = 0; i < 3; i++) {
    tråd = new Thread(new SauRunnable());
    tråd.start();
    tråd.join();
}
```

Ha heller to løkker for hhv. start() og join()

Tråder er lett å snuble i



Pass opp for deadlock (vranglås)

Hvis programmet ditt ikke ser ut til å gjøre noe som helst, men henger, så har du som oftest en deadlock-situasjon.

Det er en tråd som venter på en annen, som sannsynligvis venter på den første.

Tråder er lett å snuble i



Husk å låse

Hvis en variabel kan endres, så kan den endres samtidig av flere tråder.

Alternativt hvis du har en gruppe med data, er å dele opp f.eks. beholdere, slik at hver tråd har "sine" data alene.

Tråder er lett å snuble i



Main-tråden

Det er lett å glemme at det også er en main-tråd, som kjører uavhengig av trådene du har opprettet.

Plutselig så jobber den langt inne i programmet, eller til og med klar til å avslutte når trådene så vidt har startet.

Tråder er lett å snuble i



Race condition

Bruk signaler/condition slik at du kan kommunisere mellom trådene.

Du kan også bruke variabler, men tenk på at de må gjøres "tråd-sikre".

Da sikrer du at trådene gjør ting i riktig rekkefølge

Tråder er lett å snuble i



Synlighet

Moderne prosessorer har lokal cache per kjerne, og hvis variabelen bare er oppdatert der, vil tråder som kjører på andre kjerner ikke se endringen.

Bruk `volatile` på variabler i slike tilfeller.

Tråder er lett å snuble i



Pass opp for livelock

En annen type vranglås. Hvis en tråd venter på to eller flere hendelser eller vilkår el. for å gjøre en jobb. Da om en annen tråd også gjør det samme, så kan den ene ta den ene, og den andre den andre, og dermed ikke komme videre.

Tråder er lett å snuble i



Sult

Tildelingen av prosessor-tid til en tråd gjøres på ulike måter, og det er mulig at en tråd aldri slipper til.

Unngå `synchronized`. Prøv å opprette en lås med *fairness* satt til *true*.

```
Lock lås = new ReentrantLock(true);
```

Tråder er lett å snuble i



Ikke lås for mye

Pass på at du bare bruker lås på akkurat det som trengs. Setter du en lås på nesten hele objektet eller programmet ditt, så vil trådene bare vente. De vil ikke komme til og enten er det bare en tråd som gjør alt, eller så tar det lang tid å få noe gjort.

Tråder er lett å snuble i



Condition og signal

Husk at låser og condition henger sammen, ikke bruk condition utenfor låst region.

Bruk while-løkker når du sjekker for signaler (await()), ikke if. Spesielt hvis du bruker signalAll() istedenfor signal().

Tråder er lett å snuble i



Fil I/O

Tenk også over at fil-behandling tar veldig lang til (for prosessoren). Ofte er det ikke lurt å la flere tråder lese inn samme fil.

Det kan være nyttig å lese inn filen til minne, og så etter det la trådene jobbe på den.

Tråder er lett å snuble i



Feilsøking

Det er sjeldent at man får konkrete feilmeldinger når man jobber med tråder.

Prøv heller å tegne flyten, definere invariantene og tilstandene, og sjekk at svar er slik du forventer. Bruk forenklede datasett.

Husk at utskrifter tar tid, og gjør at tråd-timing fungerer annerledes enn når du ikke skriver noe ut. Bruk kanskje en if sjekk?